

For Reference

NOT TO BE TAKEN FROM THIS ROOM

Ex libris
UNIVERSITATIS
ALBERTAENSIS



THE UNIVERSITY OF ALBERTA

A SIMULATION STUDY OF THE CP/67
TIME-SHARING SYSTEM

by



BRIAN JOHN STANGER

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1970

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read,
and recommend to the Faculty of Graduate Studies for
acceptance, a thesis entitled A SIMULATION STUDY OF
THE CP/67 TIME-SHARING SYSTEM submitted by Brian John
Stanger in partial fulfilment of the requirements for
the degree of Master of Science.

ABSTRACT

This thesis presents a simulation study of the CP/67 time-sharing system. A brief survey is given of the development of computer software systems leading to the implementation of complex time-sharing systems such as CP/67. Time-sharing systems are discussed in terms of the problems they present in design and implementation, and the resulting need for evaluating their performance. Methods of evaluating system performance are presented, and a detailed description is given of a specific evaluation project, using simulation, which was undertaken by the author. Included is a description of the simulation model developed of CP/67, as well as a description of the measurement technique used to obtain data from the real system.

Finally, the thesis presents the results obtained from the measurement data and from exercising the simulation model under various load conditions and with alternate values of several CP/67 parameters.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Dr. G.H. Syms for his assistance and encouragement during the preparation of this thesis.

I also wish to thank Mr. D. Webster for his invaluable assistance in understanding and measuring the CP/67 system. In addition, I would like to thank my colleagues, A. Gatha and G. Neufeld, for their contribution to this research.

Finally, I wish to thank Dr. Syms, the Department of Computing Science, and the National Research Council for their financial assistance during my studies.

TABLE OF CONTENTS

	<u>Page</u>
CHAPTER I INTRODUCTION	1
1 The Development of Computers	1
1.1 Hardware Development	1
1.2 Software Development	2
1.2.1 Batch Monitors	3
1.2.2 Channels and Interrupts	4
1.2.3 Multiprogramming	6
1.2.4 Time-Sharing Systems	7
2 Thesis Outline	9
CHAPTER II TIME-SHARING SYSTEMS	10
1 The Design of Time-Sharing Systems	11
1.1 Primary Objective-Service	11
1.2 Secondary Objective-Utilization	12
1.2.1 Time and Space Management	12
2 The Implementation of Time-Sharing Systems	14
2.1 Scheduling Demands for Time	14
2.1.1 Round-Robin Algorithms	16
2.1.2 Foreground-Background Algorithms	18
2.2 Scheduling Demands for Space	19
2.2.1 Paging	20
2.2.1.1 Addressing and Virtual Addressing	21
2.2.1.2 Page Management	23
3 Conclusions	26
CHAPTER III SYSTEM EVALUATION AND MEASUREMENT	28
1 The Need for Evaluation	28

	<u>Page</u>
CHAPTER III (cont.)	
2 Methods of Evaluation	30
2.1 Theoretical Analysis	31
2.1.1 Empirical Analysis	31
2.1.2 Analytical Modeling	32
2.1.3 Simulation	34
2.2 System Measurement	36
2.2.1 What to Measure	36
2.2.1.1 Service to the User	37
2.2.1.2 Utilization of Resources	38
2.2.2 How to Measure	41
2.2.2.1 Data Collection	41
2.2.2.2 Data Reduction	43
3 An Alternative Approach to Systems Evaluation	43
4 Conclusions	44
CHAPTER IV AN EVALUATION OF THE CP/67 TIME-SHARING SYSTEM	
1 Objectives	46
2 General Description of CP/67	47
3 Previous Evaluation Studies	49
3.1 Evaluation Studies on CP/67	49
3.2 Evaluation Studies on Other Systems	51
4 Detailed Description of CP/67	53
4.1 CP/67 Virtual Environment	54
4.1.1 UTABLE	54
4.1.2 Addressing, Paging, and Virtual I/O	55
4.2 CP/67 Time-Sharing Environment	56
4.2.1 General Duties of DISPATCH	56
4.2.1.1 Time Charged	58
4.2.1.2 Next User Chosen	60
4.2.2 Additional Duties of DISPATCH	62

	<u>Page</u>
CHAPTER V MODEL AND MEASUREMENT TECHNIQUES	64
1 The Simulation Model	64
1.1 The User Model	65
1.2 The GPSS DISPATCH Model	69
2 The Measurement Technique	76
2.1 Program Logic	78
2.2 The Data Collected	84
CHAPTER VI RESULTS	88
1 Measurement Results	91
1.1 User Description	91
1.1.1 Request Size Distribution	91
1.1.2 Think Time Distribution	95
1.1.3 Interrupt Interarrival Time Distribution	99
1.1.4 Interrupt Wait Time Distribution	104
1.2 System Description	105
1.3 Discussion of Results	106
2 Simulation Results	107
2.1 Running Time for Simulation	107
2.2 Validation of the Simulation Model	108
2.2.1 Data Used for Validation	109
2.2.2 Evolution of the Simula- tion Model	114
2.2.3 Results from the Model and Validation	117
2.3 Results of Model Testing	124
2.4 Discussion	128
CHAPTER VII CONCLUSIONS	132
REFERENCES	139
APPENDIX A CP/67 VIRTUAL ENVIRONMENT	142
1 Addressing and Paging	142
2 Virtual I/O	146
APPENDIX B DETAILED DESCRIPTION OF DISPATCH	149
1 Virtual Environment	149
2 Queue Maintenance	153

	<u>Page</u>
APPENDIX C THE SIMULATION PROGRAM	157
1 Model Input	157
2 System Representation	158
3 DISPATCH parameters	159

LIST OF FIGURES

Figure		<u>Page</u>
2.1	Scheduling Algorithms	16
2.2	Round-Robin Algorithm	17
2.3	Foreground-Background Algorithm	18
2.4	Address Mappings	22
3.1	Space-Time Product	40
4.1	Time Charged by DISPATCH	59
5.1	User Model	66
5.2	Description of Model Input	67
5.3	GPSS User Model	75
6.1	Controlled Test Load Results	113
6.2	Validation Comparisons (a)	119
6.3	Validation Comparisons (b)	119
6.4	Validation Comparisons (c)	121
6.5	Validation Comparisons (d)	123
6.6	Validation Comparisons (e)	123
A.1	Addressing and Paging Tables	144
A.2	Virtual-Real I/O Blocks	147
B.1	DISPATCH - LOOP1	150
B.2	DISPATCH - LOOP2	151
B.3	Queue Management	154

LIST OF TABLES

Table		<u>Page</u>
6-1	Request Sizes	92
6-2	Request Size Distribution (a)	93
6-3	Request Size Distribution (b)	94
6-4	Think Time Distribution (a)	96
6-5	Think Time Distribution (b)	97
6-6	Think Time Distribution (c)	98
6-7	Interrupt Arrivals (a)	100
6-8	Interrupt Arrivals (b)	101
6-9	Interrupt Arrivals (c)	102
6-10	Initial Measurement Results	110
6-11	Model Results	118
6-12	Measurement Results	118

CHAPTER I

INTRODUCTION

With the advent of expensive large-scale computing systems, it becomes necessary to ask the question: is this particular computer system being used efficiently? In other words, is this computer system operating at maximum capacity, and giving the best possible response? To ask this question is to immediately suggest another one: how does one determine whether a computer is being used efficiently? Because recent development of computer hardware and software has been so rapid, the development of computer evaluation techniques has lagged far behind. The emergence of time-sharing system, in particular, has emphasized the need for better methods of system evaluation. In light of these developments, this thesis is primarily concerned with methods of measuring and evaluating the performance characteristics of computer operating systems, and how these methods can be used to improve the efficiency of computer systems.

1 The Development of Computers

1.1 Hardware Development

Until recently, the advancements of computer systems were generally described in terms of hardware development.

Computer hardware has progressed from the characteristic vacuum tubes of the first generation systems, to the transistor of the second generation, to the integrated circuits of the third. The major effect felt from this development was the greatly increased reliability of both logic and memory circuitries at a reduced cost [38].

1.2 Software Development

It was from this hardware development that a second and, to the user, a more profound effect was felt; that of software development. As computers became more reliable and less expensive, there naturally followed a corresponding increase in the number and variety of demands for their use. It is software which allows for the wide variety of applications that computers are used for today.

Software has progressed from a first generation characterized by machine language and assemblers, through a second generation of high-level language processors and system monitors, to the present generation featuring complex operating systems. Whereas first generation computer systems were primarily used for computation, the development of software has made data management an equally important application for computers. To both these areas, the introduction of

user-computer communication has also been a valuable software contribution.

The role of software in the operation of a computer system has reached a point where, today, the term "computer system" usually refers to the operating system, with little regard for the hardware of the system. Since this thesis is primarily concerned with the performance characteristics of current operating systems, it would be well to include here a short survey of the development of software operating systems. From this we can gain a better understanding of what is an "operating system" and what are its functions.

1.2.1 Batch Monitors

One of the limiting features of early computers was the amount of time necessary to set up a program for execution. As the number of users grew, the throughput of the computer became more dependent on the operators' abilities to manually load and unload programs. It appeared that many of the operator functions could be replaced by a program, and thus the development of operating systems was an obvious attempt to increase the computer throughput.

The early operating systems were quite simple since they were required to perform a well-defined set

of duties, usually in some well-defined, sequential order. They would read in a job, call in an assembler or compiler, load and execute the resulting object program, and, after execution, terminate the job, and begin the next. Although somewhat simplified, this description still applies to the batch operating system of today.

Performance in these systems, in terms of throughput, was greatly improved over those using manual loading techniques. Throughput capability was dependent on such hardware factors as the CPU (central processing unit) cycle time, memory access time, card reader speed, and line printer speed. The performance of the software, because of its limited duties, was not a significant factor in the overall system performance.

1.2.2 Channels and Interrupts

The straightforward, synchronous nature of operating systems disappeared with further advances in hardware and software. In particular, the development of the data channel, with its ability to interrupt the activity of the main processor, exerted the greatest influence on software development in this period [31].

The channel is essentially another computer, connected to the main processor computer, that is designed with

only the limited capabilities of controlling the transfer of data between the I/O (input/output) devices and main memory. The channel receives its instructions from the main processor but otherwise operates independently and concurrently with the main processor. In order to maximize the utilization of I/O devices (and of main processor and memory), the channel must signal the completion of an assigned task so that another task can be assigned to the channel. This is done by sending an interrupt to the main processor. Although the interrupt itself is a hardware feature, it is essentially a call to a software routine. This routine must first record the status of the current job in order that processing may be resumed later. The routine then interprets the cause of the interrupt, and performs the appropriate action before restoring the interrupted program to running status. The occurrence of other different hardware interrupts, while the first is still being processed, must be allowed.

As the software field gained in importance, other items besides hardware efficiency had to be considered in the definition of system performance. Particularly, as the number and complexity of the duties of software increased, the software efficiency became an important factor. For example, the more efficient the interrupt

routine was, the less time the main processor was required to wait for that I/O, and the more useful work it could accomplish.

1.2.3 Multiprogramming

The next important development in the design of operating systems was the implementation of multiprogramming. Multiprogramming is the maintenance of more than one program in an active state at one time within a single system [31]. Being active means that whenever the first program enters a wait state, another program is immediately available for execution. Thus multiprogramming allows the computer to utilize the time that is normally lost waiting for I/O by executing another program.

The responsibilities of the operating system expanded proportionally to the number of activities it had to coordinate. Maintaining a smooth flow of work through the computer required an increasing amount of system bookwork, keeping track of the current status of the programs sharing the computer and the computer resources for which the programs were competing. When there was more than one active program the resources, such as the CPU, main core storage, and secondary storage, had to be shared between the programs. If any of the

resources were not available when required by a program, that program must wait and another program must be found which could be executed with the currently available resources.

However, as well as improvements obtained from software developments, there were also considerations that reduced system performance. As their complexity increased, the operating systems used an increasing amount of processor time, overhead, and also precious space in main memory. While the objectives of multi-programming were to increase utilization of computer resources, especially processor time and storage space, the demands of the operating system itself placed these objectives in serious jeopardy.

However, at this point, system performance could still be fairly easily defined and measured in terms of system throughput. Reduced overhead was a possible area through which improved performance might be obtained, although its significance would probably be small.

1.2.4 Time-Sharing Systems

The latest significant development in operating systems, which brings us to the area of concern in this thesis, came with the advent of the time-sharing computer system. A time-sharing system can be defined as a

multiprogramming system in which many jobs are active. It is furthermore assumed in this thesis that a time-sharing system implies quick-response^{*}, on-line job entry. That is, a time-sharing system is one which has the capability of receiving requests for execution directly from terminal users, and is designed to give immediate response to these requests. It is also assumed that a time-sharing system implies multi-access job entry; that is, the system can receive job entries from several terminals at the same time.

Although beautiful in its concept, the result of enabling all users to be active and to "promise" them all dedicated service has had far reaching effects on the design of computer systems. The situation could arise that all users make a request at the same time, and there could be virtually unlimited demands for the resources of the computer system. How does the operating system hope to handle efficiently such a situation? This is the concern of this thesis.

* The term "quick-response" is used as opposed to the term "real-time". Real-time is usually applied to a situation where computation occurs at the same time as a related physical process, and where the results of the computation are used to guide this process.

2 Thesis Outline

The next chapter describes further the principles involved in the design and implementation of time-sharing systems. The concept of scheduling is introduced as the means of implementing a system designed to allocate a limited set of resources among an unlimited set of demands.

Chapter III attempts to show how the need for performance evaluation arises out of the new complexities created by these systems, and yet how these complexities in turn make it difficult to define, as well as to measure, system performance. Possible methods of evaluating a time-sharing system are discussed, including methods of obtaining measurement data from a system.

Chapters II and III are intended to be tutorial in nature. Those readers who are familiar with the principles of time-sharing systems might consider starting on Chapter IV.

Chapter IV describes a specific evaluation project undertaken by the author and compares it with previous projects of a related nature, as well as describing the time-sharing system involved (CP/67 at the University of Alberta).

Chapter V describes the technique used to obtain measurement data from the CP/67 system, as well as the simulation model developed. Chapter VI presents the results obtained from both the measurements and the model.

CHAPTER II

TIME-SHARING SYSTEMS

"Time-sharing of computer facilities has been widely acclaimed as the most significant evolutionary step that has been taken in recent years toward the development of generalized information utilities" [32]. Sackman goes on to show that time-sharing systems were an outgrowth of early real-time systems developed in the 1950's for command and control systems, specifically those connected with S.A.G.E. air defence [12]. Out of these systems grew the early time-sharing systems where many military operators at separate consols were able to simultaneously request and receive information from one central computing system.

From this grew the present day concept of a time-sharing system which provides a complete general-purpose computer facility to many on-line users at the same time. The general-purpose systems are designed to give each user full access to the computer's capabilities. The users of such a system can thus be conceived as a more or less random and changing collection of people, each entering and leaving the system independently, each using it for varying periods of time,

and most important, each working on unrelated tasks [32]. The system is general-purpose in that there is no restriction on the kind of program it can accommodate. Although there are also time-sharing systems which are designed for specific applications, the term "time-sharing system" used herein will refer specifically to general-purpose time-sharing systems.

1 The Design of Time-Sharing Systems

The best overall objective of a time-sharing system is "To provide the users with easier access to the computer, and to increase the interaction between the users and the computer".

1.1 Primary Objective - Services

Instead of this objective another, more fanciful, objective is often sought - that of providing each user with the impression that he is receiving the full, dedicated attention of the computer. To create this illusion, it is necessary to provide quick response to all users. Thus immediate response to users' requests becomes the main objective in designing time-sharing systems.

These systems, however, are designed primarily to handle conversational users requiring small amounts

of computing time. If several users simultaneously request a large amount of computation, it will put a high demand on the computer resources and quick response cannot be achieved. Thus the primary objective becomes one of providing the best possible response to all types of users, with first consideration being given to conversational users, and hoping that most users will limit the number of programs requiring large amounts of computing time.

1.2 Secondary Objective - Utilization

A secondary objective of time-sharing systems, like that of any multiprogramming system, is to obtain better utilization of the various system resources. (Multiprogramming and its objectives were introduced in Chapter I.) It is clear that the successful attainment of this secondary objective almost always influences the successful attainment of the primary one. The effectiveness of these systems in providing fast service depends in large part on the efficiency with which the resources of the computer are allocated to the individual users [5].

1.2.1 Time and Space Management

The secondary objective depends on two major factors - time and space management. Time management

means the assignment of computing, or CPU, time to the users. A time-sharing system has many users, all of which may be making a request for processor time at once and all of which are expecting immediate response. These requests must be handled efficiently, not only to achieve quick response, but also to minimize the overhead which could significantly reduce the total CPU time available for users.

Space management involves the storage of users' programs and data during execution as well as "overnight". When several users request execution of a program simultaneously it is physically impossible to keep all active programs in core at the same time. The high cost of core storage prohibits the provision of unlimited main core memories. This requires the provision of less expensive, and slower, secondary storage devices in which to store information not in immediate use. Because of the remote access implied by time-sharing systems, it is also essential that some area of secondary storage be provided for the user to file away his programs and data overnight, where he has easy access the next time he needs them.

The operating system becomes responsible for moving all information into and out of core and for keeping track of the whereabouts of all information

as it is being moved around. Again the speed and efficiency with which the system carries out its space management is a significant factor in the smooth functioning of the system.

2 The Implementation of Time-Sharing Systems

Although a computer often seems to do many things simultaneously in the human time scale, it does in fact handle all requests sequentially, and it is the purpose of the operating system to schedule the use of the computer's resources between the current requests. Procedurally, then, time-sharing systems are based around queues of demands for system resources. If a particular resource is not available, a demand for that resource must wait in queue until it becomes available. "Scheduling is the process of allocating resources to workload to satisfy some objective of good service" [16]. It is in the area of scheduling theory that the actual implementation of time-sharing systems is based.

2.1 Scheduling Demands for Time

The distinguishing feature of a time-sharing system, and the one after which it is named, is the concept of time-slicing. An ordinary multiprogramming system will normally run a program to completion, whereas

the time-sharing system will only give that program one time-slice in which to execute. At the end of that time-slice, which is some small time increment, say 50 milliseconds, a timer-controlled interrupt occurs, the program is automatically terminated, and another is started. The terminated program must wait for another time-slice to resume execution.

The theory behind time-slicing is that it allows shorter jobs to be completed more quickly by preventing long jobs from dominating the system. In essence it gives high priority to short requests, which is in accordance with the objectives of time-sharing.

Important consequences are implied by the occurrence of these timer interrupts. Each such interrupt means the system must look for another job to execute, and it is in this decision-making process that the influence of the operating system becomes evident in the overall operation and performance of the system. Various schemes, better known as job scheduling algorithms, have been developed to make this decision of which job should execute next in order to best meet the objectives of a time-sharing system.

The role of these algorithms is shown schematically in Fig. 2.1. They basically involve ordering the users who are currently requesting processor time and

and placing them in a queue. The next user to execute is selected from the front of the queue. It has been shown [35]

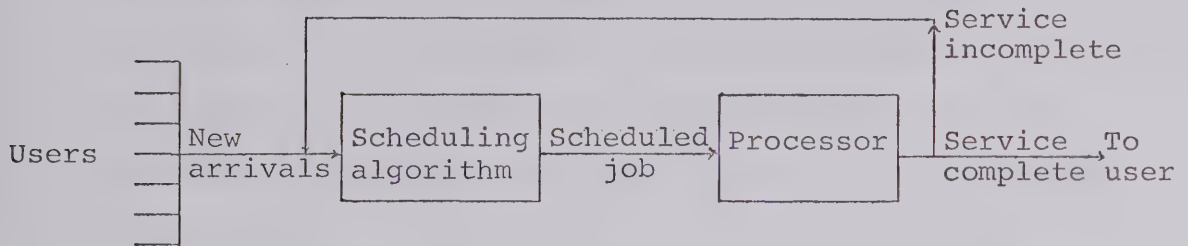


Fig. 2.1 Scheduling Algorithms

that the optimal choice of a next user is the one with the shortest remaining processing time. However this requires prior knowledge of the size of the request, which the operating system does not have. We shall next discuss some of the techniques which have been developed in an attempt to overcome, or at least minimize, this handicap.

2.1.1 Round-Robin Algorithms

The round-robin scheduler is shown schematically in Fig. 2.2. Jobs are taken from the front of the queue and provided with a time-slice of service. If the job chosen completes execution within the time interval allocated, then it is simply ejected from the system

(i.e. the results are sent back to the user at the terminal). If, on the other hand, the chosen job requires more time than allocated, it is removed from execution and put at the back of the queue. After all jobs ahead of it have also received their time-slice of service, it is again given another time-slice, and execution is started at the point where it was previously interrupted.

The decision-making process is quite simple in this case. First choice goes to the job which has waited the longest in queue (first-come, first-served). The fault of this algorithm is that large jobs which have been in the system for a long time are getting the same priority as those which have just entered and which may be quite short.

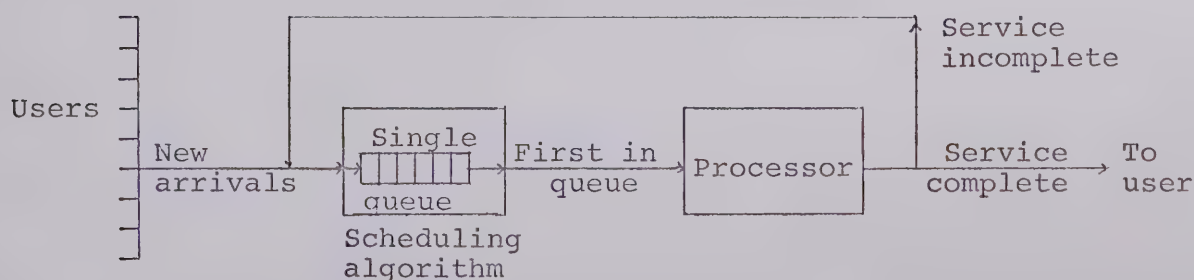
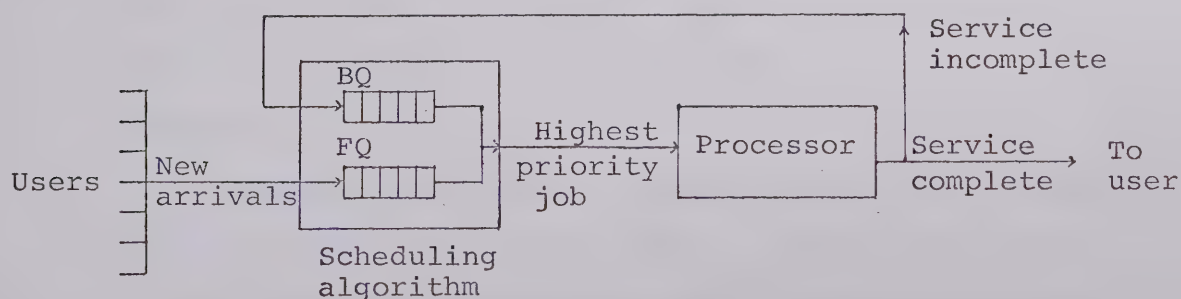


Fig. 2.2 Round-Robin Algorithm

2.1.2 Foreground-Background Algorithms

There are a multitude of scheduling schemes which fall under the heading of foreground-background algorithms. The basic implementation is shown schematically in Fig. 2.3. All jobs waiting to be executed are assigned a priority based on information taken from the previous execution of that job. New arrivals are given top priority by being placed in the foreground queue. If, after one complete time-slice, they have not completed execution, they are put in the lower priority background queue. When a new job is required for execution, the foreground queue is searched first. If there are no jobs on the foreground queue, then the first job on the background queue is given another time-slice. Within each queue, selection is first-come, first-served.



FQ - Foreground Queue

BQ - Background Queue

Fig. 2.3 Foreground-Background Algorithm

However there is a lot of variation from the basic algorithm. Instead of one background queue, there can be several. After a complete time-slice in one background queue, the job can be put at the back of the next, which is of lower priority. On the other hand, a job can be allowed to remain in a particular queue for more than one time-slice. Of the many possible variations, the job scheduler of CP/67 is an example which will be described in more detail in Chapter IV.

2.2 Scheduling Demands for Space

The problem of space management was introduced in Section 1.2.1. It was pointed out that the total core storage required when several jobs run concurrently will generally exceed the actual physical capacity of the high speed core storage. Swapping and paging are two common methods used to handle this problem of space management.

Swapping was the first method used by early time-sharing systems [34,13, 7]. When a program was chosen to run, the entire program and all its data was brought into core. When execution halted the program was restored (swapped) back onto secondary storage and a new program was brought in for execution. Since the

early systems had only a few users, the swapping method was satisfactory; but with the introduction of large time-sharing systems with many users, the shortcomings of swapping became evident. This was especially due to the great amount of time needed to swap the programs in and out of core.

A partial solution was to allow more than one program to remain in core at the same time in a multiprogramming fashion. However, as always when more capacity is available, the size of programs increased until any two programs overflowed the storage capacity, and new algorithms were required.

2.2.1 Paging

Rather than require that the entire program be resident in core during execution, one solution lies in allowing the job to execute with only part of the program and data actually residing in core. This is the principle behind paging. By breaking all information into small units of storage called pages, it is possible to keep in core only those pages of a program required to remain active. If more information is needed for continued execution, only that page which contains the needed information is brought into core.

The desired effect of paging is to optimize the number of users who can actively use core simultaneously.

This, of course, directly affects the service to the user as well as optimizing the utilization of core storage.

The implications of paging on the duties of the operating system are analogous, and in addition, to those introduced by time-slicing (see Sec. 2.1). The absence of a page implies an interruption in processing, during which the operating system must find the missing page in backup storage, decide where to put it in core, and then bring it into core. The implementation of paging has had a major effect on the operating systems involved. "In fact storage allocation has come to be regarded as one of the basic responsibilities of the computer system itself" [21]. There are two areas of responsibility involved - addressing and the moving of pages.

2.2.1.1 Addressing and Virtual Addressing

Because a program may not reside in core in its entirety, each reference to an address must now be checked to ensure that the addressed information does indeed reside in core. If it does not, then execution of that program must be suspended while the required page is moved into core. Even if the needed page is in core, the duties of the operating system in referencing

the page are not over, since the pages belonging to one program are scattered throughout core and each one must be located separately.

The operating system is thus responsible for translating all addresses referenced during execution into the real addresses of the information. This translation is done by means of a procedure which is usually partly hardware and partly software and involving a series of mappings. A simple such mapping scheme is illustrated in Fig. 2.4.

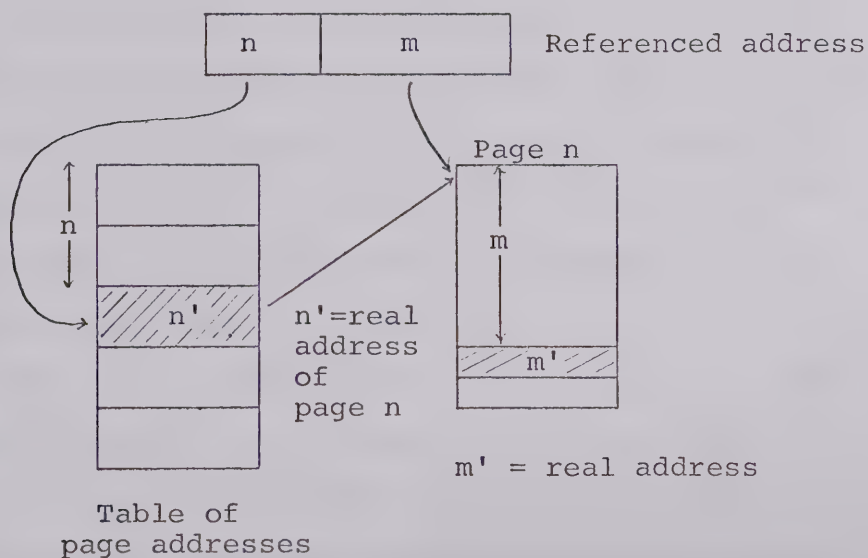


Fig. 2.4 Address Mappings

An important consequence, to the user, which arose from the addressing techniques introduced by mapping,

was the concept of virtual addressing. Because it was no longer necessary for the entire program to reside in core during execution, a program was no longer limited by the size of core memory. The task of executing a program of say 64 K words, in a memory with a size of say 32 K words, would be accomplished entirely by paging. It would be done with no assistance from the user who would normally be unaware of the size of the actual memory.

2.2.1.2 Page Management

The second area of responsibility introduced by paging was in the method used to move pages in and out of core. To move a page in, there are three obvious strategies possible. A page can be brought in before it is needed, or at the moment it is needed, or later at the convenience of the system.

The first method is most ideal in that it would prevent a program from waiting for missing pages. However it is also the most impractical in that it requires prior knowledge of which pages the program is going to need. Such information is generally not available, although research [14] has been carried out to determine if any patterns exist. Research is also being conducted in the third approach [4, 9] which

is probably the most practical paging strategy because it enables better utilization of I/O devices.

However, the second method, which is called demand paging, has been used almost exclusively to date. A page is brought into core only on demand - at the moment it is needed. Demand paging has the desired effect of minimizing the amount of core storage allocated to each program. Since this is the method used by CP/67, it will be discussed further in Chapter IV.

Whatever the method selected however, the real problem is not in deciding which page to bring in, but in deciding which page to remove in order to make room for the new page. The best decision would be to remove the page with the least likelihood of being needed in the immediate future [10]. Again various schemes, better known as page replacement algorithms, have been developed to make these decisions. Their situation is analogous to that of the scheduling algorithm's in that they are required to make decisions about the future paging requirements using knowledge only of the past. Yet their decision can have a significant effect on the page traffic^{*} and hence on

* page traffic - the number of pages per unit time being moved between memories [10].

the performance of the system (see Chapter III, Sec. 2.2.1.2). Any decision to remove a page which is going to be needed immediately is obviously a bad decision.

Two of the basic techniques used by these algorithms are outlined as follows: the First-in, First-out (FIFO) algorithm says whenever a fresh page is needed, the page which currently has resided in core the longest is chosen to be moved out. The Least Recently Used (LRU) algorithm says whenever a fresh page is needed, the page which has remained unreferenced for the longest time is chosen to be moved out.

Again there are variations and other considerations such as whether or not the page chosen actually needs to be rewritten into secondary storage. If it has not been altered in any way while in core, then it's original image still remains in secondary storage and there is no need to move it back (providing a record is kept on which pages have been altered). Thus choosing an unaltered page will reduce the page swapping time to a half.

The responsibility of the page replacement algorithm becomes evident when one considers how as the number of users in core increases, the more paging is required; and

as more paging is required, the longer users must occupy core (waiting for pages), and the more crowded it becomes The situation then arises where the system is doing little else except paging. All users will be waiting for pages and no useful work will be accomplished. Such a condition is called thrashing and is discussed in more detail by Denning [11].

3 Conclusion

Time-sharing systems are primarily an attempt to provide easy access and good service to the computer user. A secondary objective is to achieve better utilization of the computers resources. Both these objectives can be met by allowing many users access to the computer concurrently. Conflicting demands for the use of computer resources are handled by scheduling techniques.

Although actual implementation of such systems have shown their potential, their performance under heavy load is still questionable. Although some limit is expected as to the load such systems can efficiently handle, the actual limit is quite often lower than expected.

Performance in these systems is a measure made up of many interactive variables. The effectiveness of the scheduling algorithm, the effectiveness of the page replacement algorithm, the amount of overhead involved in each of these, and the time required to transfer a page are some of these variables. Each contributes to the length of time a job remains active in order to complete its execution. The longer a job is active, the longer the response time to the user, and the greater the probability that another job will require the same resources, resulting in further delays and congestion.

If performance is found to be unsatisfactory, the question arises as to which of these variables cause the poor performance. In the next chapter we investigate methods of measuring and evaluating time-sharing systems in order to analyze their performance.

CHAPTER III

SYSTEM EVALUATION AND MEASUREMENT

1 The Need for Evaluation

Progress in designing and applying information handling systems has outraced progress in evaluating their performance [2].

Within the past several years, technology has outstripped methodology in the evaluation of large information processing systems [36].

Although these statements are now three years old, they reflect a situation still existing within the computer industry which was brought about by the development of time-sharing software systems. These new computer systems have produced a significant and somewhat unanticipated increase in complexity. Methods of evaluating system performance progressed too slowly, with the result that many of the new systems were developed and implemented without the assistance of proper analysis.

The changes introduced by these systems were the result of three closely related factors. The first was the influence of the hardware configuration on the operation and performance of the system. Formerly hardware systems were usually obtained as a sort of "package deal", with different packages available for

different applications or size of operation. However time-sharing systems are more frequently "tailor made", using a variety of possible components. Size and speed of core; number of drums; number, size and speed of disks; number of data channels are some of the variables to consider in "building" a time-sharing computer system.

The second factor was the increased role of the software (operating system) in the total operation, and hence performance, of the system. Software components such as the interrupt handling routines, the scheduling algorithm, and the space management routines have a substantial effect on the overall performance of a system.

The third, and perhaps most important, factor was the great interdependence of the various software and hardware components just mentioned. The effectiveness of the software depended on the efficiency of the hardware, and vice versa. The total result of these changes was a broadening (or obscuring perhaps) of the definition of system performance. This chapter will discuss methods of defining, measuring, and evaluating system performance.

2 Methods of Evaluation

The evaluation of time-sharing systems is far from being a science. Good systems evaluation involves a great deal of ingenuity, guess-work, and common sense, with the biggest assets being experience and general knowledge in the field of computer systems. As far as actual methodology, the techniques used generally fall into three categories:

- a) empirical analysis
- b) analytical modeling
- c) simulation.

These three methods are not distinct and might better be thought of as successive levels in evaluating computer systems. The level used can be chosen to fit the level of understanding of the system and the degree of analysis desired.

Before going on, it is important to emphasize that implicit behind all these methods is the necessity for measurement of the real system under study. Measurement involves the gathering of quantitative data on the behavior of that system. As Cantrell states; "All good analysis consists of a combination of theoretical analysis and empirical measurement" [3]. Measurement is needed to give relevance to the theore-

tical analysis. The question of how to obtain measurements of a system's behavior is discussed in Sec. 2.2.

2.1 Theoretical Analysis

2.1.1 Empirical Analysis

Although the terms "theoretical" and "empirical" seem somewhat contradictory, they are used together in the sense that theoretical analysis is achieved simply through an understanding of how a system is supposed to work. Empirical analysis is the method of examining the quantitative data obtained from measurement and deducing the systems performance from this. (Examination of the data includes data compilation and analysis, since the raw data is seldom in a comprehensive form suitable for immediate "empirical" analysis.) The depth of this examination may vary depending on the researcher's degree of understanding of the system, the object of the research, and the content value of the data within this objective. Simply by examining the measurement results, then, it may be possible to discover problem areas within the system's performance.

The one basic drawback to this method is that it has very limited predictive powers. Any decisions made must be based on the collected data. To actually test any proposed improvements, they must be implemented and

new data obtained for examination. Also it is usually difficult to predict the performance of the system under a different workload using this method. This is particularly true for time-sharing systems, since their complex interactive nature is very unpredictable under changing load conditions. The value of empirical analysis, then, is limited to immediate performance problems.

2.1.2 Analytical Modeling

Analytical modeling is an attempt to analyze the behavior of a computer system from a mathematical viewpoint. Because the operation of time-sharing systems is centered around the principles of scheduling and queues, it can often be described, and analyzed, within the boundaries of queueing theory. The method involves describing the system in mathematical terms and applying queueing theory to this description. The results of the queueing analysis can then be taken as an indication of the behavior of the real system. The effect is, essentially, a mathematical simulation of the system.

More specifically, the description of a system is usually subdivided into two parts - one, the demands or workload entering the system and two, the actions of the

system on the workload. The demands are defined in terms of:

- a) their rate of arrival
- b) their size.

For example, a demand may be a request for processor time or a request for core space. The rate of arrival and size can be described mathematically as, say, an exponential distribution. Using queueing theory, results can be obtained such as the average response time or the average waiting time in a particular queue.

Analytical modeling, then, is useful for gaining insight into the work flow pattern through the various queues of a system and it can help to uncover areas which are reducing the smooth flow of work. It contains more predictive capabilities than empirical analysis since possible workloads can be examined by merely re-defining the demands. Hence it is easier to examine the system behavior under varying load conditions.

The disadvantage of analytic models is in their limited applicability. They are limited by the assumptions that the system behavior must follow the rigid laws of mathematics and queueing theory. As such, analytic models are generally useful only for analysing parts of a system and it is usually not possible to

combine these parts to analyse the whole system. As a result, they do not adequately reflect the interaction between the various components of a time-sharing system which, as we have stated, has such a significant effect on their overall performance.

2.1.3 Simulation

Rather than defining a mathematical model of a system, simulation is an attempt to build a logical model of the system. Simulation is generally considered as a last resort when the system becomes too complex for an analytic model. It is considered as a last resort because it requires even more work and understanding than is needed by other methods. The biggest disadvantage of simulation is the amount of time and effort needed to define a logical model of a complex computer system. However, if carefully done, simulation can bring results where other methods fail.

As before, the construction of a simulation model involves both a description of the workload entering the system and a description of the actions of the system on this workload. The workload is again defined in terms of the size of the demand and their rate of arrival. However, the actions of the system are defined logically rather than mathematically, and this is where

crucial difference lies. Simulation considers time as one of the key variables and treats each action as a discrete event occurring in a time sequence. Hence it is able to consider changes of state as a function of time and it can make logical decisions based on the present state of the system. For this reason simulation is able to model the dynamic, interactive influences involved, and to give a more complete representation of the total system behavior.

Simulation is also more easily parameterized. Because of its logical base, it is able to include a wider variety of conditions which are not well behaved mathematically. This makes it easier to include in the model any proposed changes to the real system, and hence to study their effect on the system's behavior. A good example of simulation's versatility is its ability to characterize an individual user or class of users in a time-sharing system.

Because simulation is more complete in its representation of the system, it forces the analyst to obtain a greater degree of understanding of all aspects of the system. And because it generally includes a wider range of variables in representing the system, simulation will usually require a wider range of measurements from the real system, again requiring a greater degree of detailed

knowledge. Simulation also includes the problem of writing a program to represent the model, and although several simulation languages have been developed, the programming and the verifying of the model's accuracy is a tedious task.

2.2 System Measurement

In the empirical analysis of a system, the value of system measurement is obvious. The data collected is used as a direct source for pinpointing problem areas. Regardless of whether the data can uncover ways of improving the performance, it can still be useful in indicating areas in which further analysis is required. In the latter case, the value of system measurement becomes twofold - first, it is used to obtain relevant parameters for a model design, and secondly, the data is later used to test the validity of the model.

2.2.1 What to Measure

The immediate question is what to measure. In a time-sharing system there are a large number of variables involved in the total operation of the system, and thus a first step in analysing these systems is to identify the variables which influence the system's performance. Since the importance of

each variable depends to a great extent upon the objectives of the study, there is no straightforward way of selecting the important variables.

The next consideration is the selection of a criterion for measuring these variables quantitatively. This section discusses system performance in more detail, and examines the possible criteria for measuring the performance. Following this we will discuss methods of actually obtaining such measurements (Sec. 2.2.2).

2.2.1.1 Service to the User

Since a time-sharing system is primarily concerned with providing fast, if not immediate, service to the requests of its users, it follows that the primary performance measurement will be how quickly the system responds to the users' requests. (The response time was described previously as the time between the entry of a user's request for service and the completion of that request as seen by the user.) However the measurement of response time, per se, without considering the nature of the request is not very meaningful. By measuring response time as a function of the size of the request, it is possible to relate actual response time to that expected by the user and thus establish

different performance criteria for different requests. In particular, short "conversational" requests should be measured separately from the longer "computing" requests.

2.2.1.2 Utilization of Resources

As stated in Chapter II, Sec. 1.2, the secondary performance criterion is the utilization of computer resources, especially time and space. In a time-sharing system there is a wide variety of specific measurements which might yield a suitable measure for this criterion.

First, if we are concerned with how the system allocates its processor time, the obvious measure to use is the CPU utilization. Again this measure is not very meaningful unless it is given in relation to the total amount of processor time requested by the users. Also, since the CPU utilization includes "problem time" and overhead, it is important to measure the different types of CPU utilization.

Second, if we are concerned with how the system allocates its storage space, then our biggest concern is to obtain a measure of the efficiency of the system in moving information between core and secondary storage. Such a measure is often difficult to define, particularly on those systems which implement paging in their

storage management. On such systems efficient storage management not only involves the efficient transferring of information between storage areas, but also the minimizing of the number of necessary transfers.

A suitable measure of the efficiency of information transfers is probably the traffic at the channels. With analysis of this information we can often obtain a measure of the amount of time a storage demand might expect to wait unnecessarily while the channel is busy. If this expected time can be lessened, then hopefully we are achieving better utilization of storage facilities.

In a paging system another good measure of efficiency of space utilization is simply the page traffic - or, in other words, how well the operating system keeps the number of page transfers to a minimum. Or, as McCredie [23] suggests, a proper evaluation of a given page replacement algorithm should consider both the overhead necessary to process a page as well as the number of these faults. Excess overhead in executing the algorithm will override any advantages obtained by making a "good" decision.

Kuehner and Randell[21] consider that the time required to fetch a page is the critical factor in the performance of a paging system. They suggest, as does

Joseph [19], that the "space-time product" is a good measure of a paging algorithm's effectiveness. The space-time product (see Fig. 3.1) is the product of the size of an active program in core and the time it spends in core. The size of the program will vary according to the number of pages it has in core at any particular time. If the time required to fetch a page is large, it can be seen that a large proportion of the space-time product will represent the amount of time the program occupies working storage but is inactive waiting for the arrival of another page. This measure is very useful in indicating the improvement in performance that could be achieved by lowering the page-wait time.

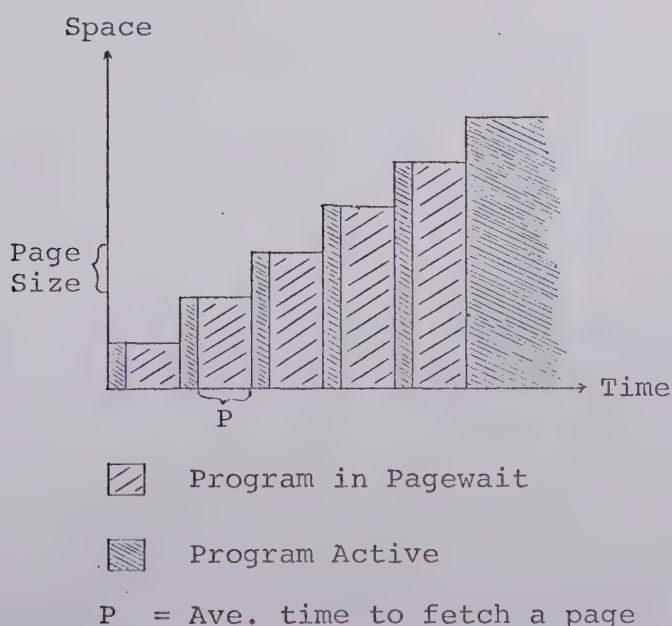


Fig. 3.1 Space-Time Product

Thus there is a wide variety of measurements that can be used to indicate the efficiency of resource utilization. We have suggested only a few which are generally considered to be the basic performance measures. It must also be realized that all these measurements are a function of other variables, particularly the load on the system.

2.2.2 How to Measure

After deciding what measurements should be made, the problem becomes how to obtain these figures. Indeed the greatest percentage of time in a system's evaluation project is usually spent in the collection and reduction of real world data.

The importance of the data gathering techniques in the overall planning of a project must be emphasized. The questions of what to measure and how to measure it are by no means distinct decisions. Often the exact figures wanted are not available within the means of data collection and, as a result, alternative figures have to be obtained. If relevant data cannot be obtained, the feasibility of the entire project or the validity of its results are in jeopardy.

2.2.2.1 Data Collection

Measurement techniques can be divided into two categories:

- a) hardware
- b) software techniques.

Hardware measurement involves the use of a hardware device to record activity within the computer. The big advantage of this technique is that it imposes no interference on the system it is measuring. Also the figures obtained are very precise. This is especially valuable if essential measurements involve small time intervals, which is usually the case. The big disadvantage of hardware measurement is in the time and knowledge needed to initially develop the necessary device. Also hardware measurement is limited to measuring hardware features such as a CPU in wait state, or a channel busy, etc., and will not measure many software features. Shulman [36] and Roek [30] discuss specific applications of hardware measurement devices.

Software measurement involves the modification of the existing operating system to extract information during the operation of the system. The disadvantage, of course, is that this uses CPU time and storage space, and therefore interferes with the normal operation of the system. The advantage of software measurement is that it allows access to information on the functioning of the software

components such as operating system overhead, etc. Also the range of information obtainable is greater since software is generally easier to modify than hardware.

Deniston [8] and Pinkerton [29] describe the use of two specific software measurement applications. A unique software technique, which was used to obtain data from the CP/67 time-sharing system, will be discussed in Chapter V.

2.2.2.2 Data Reduction

Regardless of the measurement technique used, the next problem is to reduce or condense the raw data. Information from the data must be converted into a meaningful format for the human analyst. This can often involve developing quite complex programs capable of recognizing hidden events or behavior patterns of significance. Only then can the theoretical analysis of the data begin.

3 An Alternative Approach to Systems Evaluation

Karush [20] presents an alternative approach to the empirical measurement-theoretical analysis philosophy presented here. His method is analogous to the benchmark techniques used to evaluate non-time-

sharing systems. It involves the development of "stimulus" programs which are designed to test the system under actual operating conditions by simply executing these programs. Results are obtained merely by measuring the external response of the system to these stimuli.

The advantage of this stimulus approach is that results are obtained easily and immediately. There is little data reduction necessary nor does it involve the complexities of getting "into" a system to obtain the data or the degree of understanding needed for theoretical analysis. The disadvantage of this approach is again that the range of measured performance is limited to the actual computer operation, and it is not easy to experiment with alternative system configurations. Also, it does not provide any data on the secondary performance criteria (i.e. resource utilization) which could be helpful in improving the overall performance.

4 Conclusions

Since all methods of evaluating a computer system seem to have some disadvantages, many factors must be considered in making a choice of the best approach. The objectives of the evaluation project (i.e. the

degree of analysis desired) is, of course, the most important factor. Others which can have a significant influence include knowledge of the system, the measurement techniques available, the data available, time available, as well as experience in computer systems generally.

In the next chapter, we begin a description of a specific project undertaken by the author involving the evaluation of a time-sharing system.

CHAPTER IV

AN EVALUATION OF THE CP/67 TIME-SHARING SYSTEM

1 Objectives

The first obvious step in undertaking any evaluation project is to set down some objectives for the study. The project presented here was centered around an evaluation, using simulation, of the job scheduling algorithm of the CP/67 time-sharing system. The primary object of the study was to investigate the relationships between the performance of the system and various parameters of the scheduling algorithm. Inherent in this objective was the idea that the simulation model could be used to find areas where present system performance could be improved by changing certain parameters, or perhaps even altering the design of the scheduling routine. The system performance under various load conditions was also investigated.

More specifically, this project is concerned with analysing the CP/67 system performance as it pertains to the management of its time resource. (The second aspect of system performance, space management, is allowed for but not dealt with in detail.) The two measures

used are the response time and the CPU utilization, both of which fall under the responsibility of the scheduling algorithm. Response time is measured in relation to the size of the request and the load on the system. The load on the system is defined solely by the number and type of users on the system. CPU utilization is obtained by examining variations in the amount of time spent by the processor in wait state, supervisor state, or problem state. The ratio of time spent in supervisor state to time spent in problem state is one measure in which improvement is expected.

Parameters of the scheduler which may be altered include the length of the time-slice, the maximum queue lengths allowed, and the quantum (or time-limit) allowed in each queue. The overall objective is to alter these parameters on a simulation model and thus determine their effect on the performance of the CP/67 system under various load conditions. The results would be useful in improving the operation of CP/67.

2 General Description of CP/67

The particular time-sharing system involved in this project was the CP/67 installation at the University of Alberta (U of A). CP/67 is a control program designed by IBM at Cambridge Research Center for implementation

on an IBM System/360, Model 67. It is designed to provide a time-sharing environment in which each user believes that he has the complete resources of the computer dedicated to his use. It achieves this objective by generating a "virtual computer" for each active user and then by sharing the resources of the real computer among these virtual computers.

The concept of virtual computers is closely related to that of virtual addressing, which was introduced in Chapter II. Virtual computers function like real ones but are the product of software simulation. Under CP/67, the user is initially assigned a virtual computer with the configuration of his choice and a CP/67 identification number. Different users may have different configurations depending on their needs. When a user signs on, CP/67 creates a virtual computer of his predefined configuration. To the user, his virtual machine appears as a real computer and he uses it as if he were at the main console of a real computer.

The user is able to equip his virtual machine with any operating system that will run on an IBM 360/65 and give him the desired functional capabilities. In most cases CMS (Cambridge Monitor System) is used, although other supervisors such as OS and APL, under DOS, are also used. CMS is a single-user, conversational

operating system designed especially to operate under CP/67. CMS gives the user the ability to initiate, monitor, and terminate his program by carrying on a command-response type dialogue with the (CMS) system.

3 Previous Evaluation Studies

3.1 Evaluation Studies on CP/67

Of particular interest here is an earlier evaluation study carried out by A. Gatha [15] on the same CP/67 installation at the U of A. This study was of the empirical analysis type in which data on the system was collected and analyzed statistically to evaluate the system performance. One of the most important results of Gatha's work was the method developed for collecting data from CP/67. His method was extended by the author for this study and is discussed in detail in Chapter V.

In brief summary, Gatha's results show that the paging load (total number of pages transferred in or out of core) increases exponentially with the number of users on the system. From his data, he calculates that with 16 users the paging (disk) channel capacity would be exceeded. He also concludes from his data that it would be unrealistic to expect the average time spent in problem mode to be much greater than 75%.

That is, when the system is loaded, at least 25% of the CPU time would be necessary for supervisory duties (i.e. overhead). At this rate, he conjectured that maximum CPU utilization would be reached with 21 users. However the limiting factor in CP/67's user capacity, according to Gatha's evaluation, would appear to be in the paging capacity of the disk channel.

Gatha's results must be qualified if they are to be compared with the results obtained in this study. At the time of his research, CP/67 paging was handled with disk storage only, whereas at the time of this research it was handled with drum storage, with disk being used only as backup to the drum*. Also, the size of core memory increased from 512 K to 768 K during the intervening time.

Another evaluation study of CP/67 at the U of A was carried out by O.S. Bishop [1] using the analytical modeling approach. He developed a priority queueing model in which system interrupts were treated as arriving from a preemptive queue with top priority. User requests were treated as arriving from two non-preemptive queues with a foreground-background type priority scheme. All three queues were, of course, competing for service from a single CPU.

* The transfer rate from the drum is 4 times faster than the transfer rate from the disk.

However, because he had no measurement data from which to accurately describe the input to his model, his results do not describe the performance of the existing system nor indicate its capacity.

3.2 Evaluation Studies on Other Systems

The results of some other evaluation studies of different time-sharing systems are of interest. Scherr [33], for example, made a study of Project MAC's Compatible Time-Sharing System (CTSS) using both analytical and simulation models. The primary measure of system performance was the response time. Because of the accuracy of his Markov model in predicting the performance of the system, even when it was a highly simplified version, Scherr concludes that performance is determined in large part by only the mean interarrival time of requests, the mean request size, and the number of users signed on. The results of his simulation studies also point out that good accuracy can be obtained from a fairly simple model. CTSS, however, does not implement paging so his results cannot be compared directly with the ones from this study.

Oppenheimer and Weizer [28] carried out a simulation study on the RCA Spectra 70/46 Time-Sharing Operating

System (TSOS). They also used response time as a primary measure of system performance. TSOS uses paging. They conclude from their results that secondary storage should be a drum or non-moving arm disk, since any slower device becomes a bottleneck for the entire system and significantly lowers the system performance. They also suggest that a mechanism should be provided for continuously monitoring and adjusting the load on critical system resources, such as I/O channels and memory, to prevent overloading. Since their results also showed that very few requests required a full time-slice before being halted by an interrupt, or by completion of the request, they conclude that the size of the time-slice is not critical. As a result, they settled on a time-slice of 2 seconds for use in the system. The main value of the time-slice is that it prevents long tasks from monopolizing the processor.

Nielsen [26] performed a simulation study on the IBM 360/67 Time-Sharing System (TSS, version 1). Since simulating the paging characteristics of a time-sharing system is very complex, Nielsen devotes another article [27] on his approach to this problem. Using the CPU utilization and response time as the measures of system performance, Nielsen also pinpointed the major source of poor performance to the bottleneck created by slow secondary storage devices used in paging.

All authors agreed that their results demonstrated the ability of simulation to provide valuable insights into the characteristics of complex time-sharing systems.

4 Detailed Description of CP/67

CP/67 is responsible for creating a time-shared, virtual environment. It accomplishes this by [6]:

- a) scheduling and allocating main storage space, CPU time, and I/O devices to the virtual computers;
- b) handling all interrupts;
- c) protecting system files, users' programs, and users' data during execution; and
- d) keeping statistics on the use and performance of the "real" system.

In this project we are primarily concerned with the CP/67 job scheduler and how it creates the time-shared environment (Sec. 4.2). However the implementation of virtual machines will be discussed first, since some knowledge of the total responsibilities of CP/67 is necessary for a better understanding of the duties of the scheduler.

4.1 CP/67 Virtual Environment

As suggested, virtual computers are the product of software simulation. It is the duty of the control program, in this case CP/67, to translate all virtual references into their real equivalent. This translation process is carried out through a series of mappings between a set of tables maintained by the control program. A complete, and dynamic, description of each virtual computer is maintained in these tables. To understand the virtual environment of CP/67 for our purposes, it will be sufficient to describe some of the tables and mappings involved.

4.1.1 UTABLE

For each user which is signed on to CP/67, there exists a primary user control block called the UTABLE. Along with the virtual I/O blocks, the UTABLE completely reflects the status of the virtual machine. The UTABLE is a starting point from which CP/67 is able to reference any information it needs.

Some of the information included in the UTABLE is:

- a) VPSW - the user's virtual PSW (program status word);
- b) SEGTABLE - pointer to the user's segment table;

- c) VMSTATUS - a byte reflecting the state of the virtual machine, where
 - X'80' indicates PAGEWAIT,
 - X'40' indicates IOWAIT, and
 - X'20' indicates CFWAIT;
- d) VTOTTIME - time used by virtual machine in problem mode since signed on;
- e) TIMEUSED - time used charged to virtual machine since signed on (includes VTOTTIME plus CP/67 overhead);
- f) NEXTUSER - pointer to next user's UTABLE;
- g) USERID - user's CP/67 identification;
- h) PENDING - contains a bit for each selector channel which has a pending interrupt;
- i) CPRREQUEST - pointer to stack of control program execution request blocks;
- j) VMXSTART - pointer to the first virtual device block on the virtual multiplexor channel;
- k) VCHSTART - pointer to the first selector channel block.

4.1.2 Addressing, Paging, and Virtual I/O

The techniques used by CP/67 to reference information are outlined in detail in Appendix A. Again they basically involve the use of various tables and mappings

by which the system is able to find the real location of information referenced by the virtual machine.

Fig. 2.4 (Chapter II) illustrates a simple example of how virtual information is addressed. If a page is found not to be in core when needed, similar mappings locate the page in secondary storage and the page is read in. Similarly for virtual I/O, all references to virtual devices are translated, via tables, into their real equivalent. When the real I/O is completed this process is reversed and the real operation is reflected back to the virtual machine.

4.2 CP/67 Time-Sharing Environment

4.2.1 General Duties of DISPATCH

When all interrupt handling routines in CP/67 complete their processing, they transfer control to the main job scheduler and control routine, which is called DISPATCH. The duties of DISPATCH are:

- a) to charge all time used to the appropriate user(s); and
- b) to determine the next user to receive control and to pass control to him.

To prevent overloading, DISPATCH allows only a subset of all users to be active at any given time. DISPATCH maintains two queues of active users, both of

which have a maximum limit on their size. One queue (Q_1) is maintained for interactive users, whose requests are characteristically small, and another queue (Q_2) is maintained for those users whose requests are for larger amounts of processor time. If a user desires to enter a queue (i.e. become active) and that queue is full, he must wait until another user leaves that queue.

A user is in one of the following five states at any given time:

- a) waiting to enter Q_1 (in Q_1 -WAIT);
- b) in Q_1 ;
- c) waiting to enter Q_2 (in Q_2 -WAIT);
- d) in Q_2 ; or
- e) inactive, not requiring system resources.

In addition, an active user may or may not be runnable.

A user is unrunnable if he is:

- a) in PAGEWAIT - waiting for a page to be brought in;
- b) in IOWAIT - waiting for the initiation of some I/O operation;
- c) in CFWAIT - waiting for a response from the terminal; or
- d) in WAIT - the virtual machine is waiting for work.

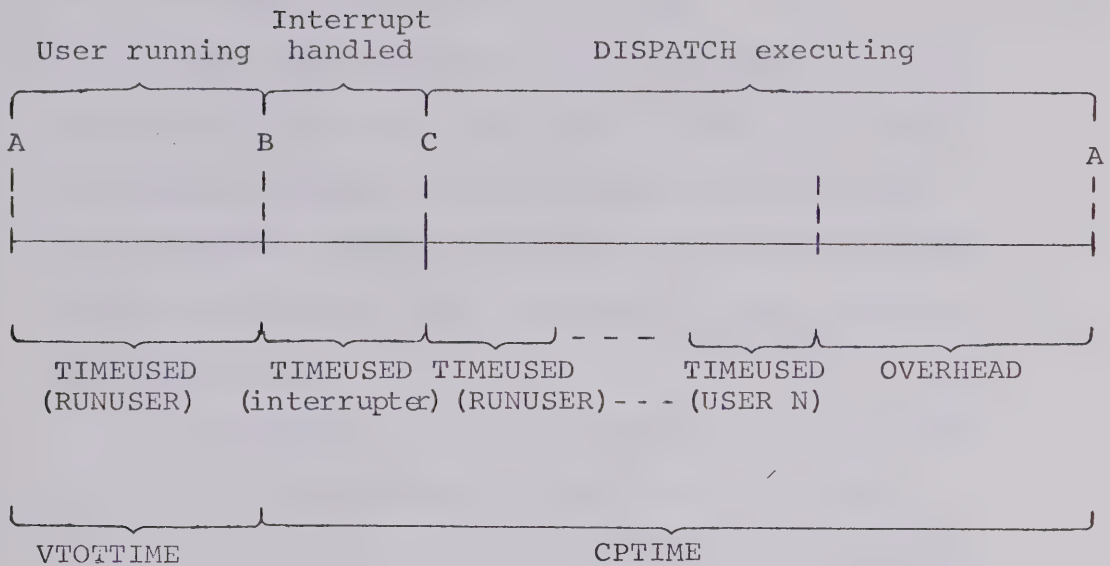
Note that an unrunnable user is still active (i.e. a member of a queue), although the WAIT condition also occurs when the user is inactive.

4.2.1.1 Time Charged

Each time DISPATCH is entered, and during its own execution, it is responsible for charging all time used to the appropriate users. There are two exceptions. The time used by DISPATCH in choosing a new user is not charged to any specific user, but is charged to the system as OVERHEAD. Similarly, time spent by the system in an idle state is charged to the system as WAITTIME.

Fig. 4.1 illustrates the time charging scheme used by DISPATCH. More detail is given in Appendix B.

As soon as DISPATCH is entered, it charges the time used to handle the current interrupt to the user responsible, and charges the interrupted user (RUNUSER) for the time used by him before he was interrupted. As well as being added to the TIMEUSED for RUNUSER, this second charge is also added to the user's VTOTTIME (the amount of his TIMEUSED actually spent in execution). The rest of his TIMEUSED arises from the time spent by CP/67 in maintaining his virtual environment (i.e. translating his virtual interrupts into real interrupts and vice versa).



Time A - user given control

B - interrupt occurs

C - DISPATCH entered

N - no. of users signed on

Fig. 4.1 Time Charged by DISPATCH

The amount of time spent in the supervisory state is accumulated as CPTIME. This is the time that CP/67 itself has control of the CPU and can be considered as the overhead of the system (not to be confused with OVERHEAD as defined above).

4.2.1.2 Next User Chosen

The second responsibility of DISPATCH is, of course, to select the next user to run. All users are chained together via the NEXTUSER pointer in their UTABLE. DISPATCH searches each user on this chain and selects a user according to the following six choices:

- 1) RUNUSER, if still runnable, and if he has not completed his time-slice, is given immediate control for the unused remainder of his 50 millisecond (msec.) time-slice;
- 2) first NEXTUSER who is in Q_1 and is runnable;
- 3) first NEXTUSER who is in Q_1 -WAIT and is runnable;
- 4) first NEXTUSER who is in Q_2 -WAIT, is runnable, and has the highest priority (see item (f) below) of all other runnable users waiting to enter Q_2 ;
- 5) first NEXTUSER who is in Q_2 , is runnable, has less than one NOQUANT (see item (e) below), and has the highest priority of all other users in this same class (CLASS 1);
- 6) first NEXTUSER who is in Q_2 , is runnable, has at least one NOQUANT, and has the highest priority of all other users in this same class (CLASS 2).

If DISPATCH is unable to find a user to run, it puts CP/67 into a wait state for an idle cycle of .25 seconds (sec.).

There are several important items to note here.

- a) Choices 2-6 are all given a time-slice of 50 msec.
- b) Choices 1-2 are given control as soon as they are found, whereas choices 3-6 are given control only after all users have been examined once (more detail in Appendix B).
- c) Choices 3-4 are made on the condition that the respective queues are not already full.
- d) Users entering Q_1 are placed at the back of the queue, behind those already in Q_1 . Users entering Q_2 are placed at the front of the queue, ahead of those already in Q_2 .
- e) Q_2 is divided into two internal queues, which we will call classes, depending on whether or not the user has one or more NOQUANT. When a user first enters Q_2 , his NOQUANT is set to zero (i.e. he always enters CLASS 1 first). It is incremented by one each time he uses a full 50 msec. time-slice without an interruption. A user in Q_2 will remain in CLASS 1 as long as he never uses a full time-slice.

- f) Q_2 has an internal priority scheme, covering both classes, whereas users in Q_1 are chosen strictly on a FIFO basis (see item (g) below). Q_2 is FIFO within each priority level. The priority scheme in Q_2 is based on the number of 5 sec. quanta that the user has spent in Q_2 . Each time a user completes 5 sec. of TIMEUSED in Q_2 , his priority is lowered.
- g) Since DISPATCH begins each search with RUNUSER (who generally is a different user each time), and although users are selected FIFO within a queue, their position in queue depends on their relative position to RUNUSER on the NEXTUSER chain. This introduces a random factor to the ordering of the queues.

4.2.2 Additional Duties of DISPATCH

In addition to the two general duties described above, DISPATCH is also responsible for updating the user queues and for maintaining the virtual environment of each user. These duties are described in detail in Appendix B. Updating the user queues basically involves examining each user and placing him in his proper state (Q_1 , Q_1 -WAIT, Q_2 , Q_2 -WAIT) if he is active. The maintenance of the virtual environment involves examining

each user's UTABLE for the presence of CPRQUESTs and PENDING interrupts. Both essentially signify that some real interrupt has occurred which affects the status of the virtual user and, if they are found, DISPATCH must reflect this change in status to the virtual machine.

CHAPTER V

MODEL AND MEASUREMENT TECHNIQUES

1 The Simulation Model

Simulation involves setting up a model of the real situation, or system, and then performing experiments on this model [24]. A computer simulation model is a logical-mathematical representation of the real system which is programmed to be run on a computer [22].

Constructing the model of any system involves two basic descriptions. The first is a description of the input to that system, and the second is a description of the actions of that system upon its input. In a computer simulation model, the system input is represented mathematically, while the actions of the system are represented logically. Thus a program is written which will accept the mathematical input and perform the logical operations of the system.

The real system being simulated in this project is the job scheduler (DISPATCH) of the CP/67 time-sharing system. The CP/67 environment consists of several users at remote terminals, each working independently on individual tasks. The model will consist

of a description of individual users, in terms of the input they represent to the system, as well as a description of how DISPATCH handles this input.* The next two sections will explain the model in more detail.

1.1 The User Model

From the user's point of view, the services of the CP/67 system are totally dedicated to his individual use. To the user the system is in one of two states; either the system is waiting for the user to enter a request, or it is working on a request of that user and the user is waiting for the system to respond (see Fig. 5.1).

Since a major portion of the time involved in entering a request is taken by the user in merely thinking about his next request, this period of time will be referred to as the "think time". The think time also includes the time involved in typing the request and the time to receive the response output. The response time begins when the user enters a request and ends when the response begins to be sent back to the terminal. As well as the time spent actually

* These two parts could be referred to as models of the (system) input environment and of the system operation.

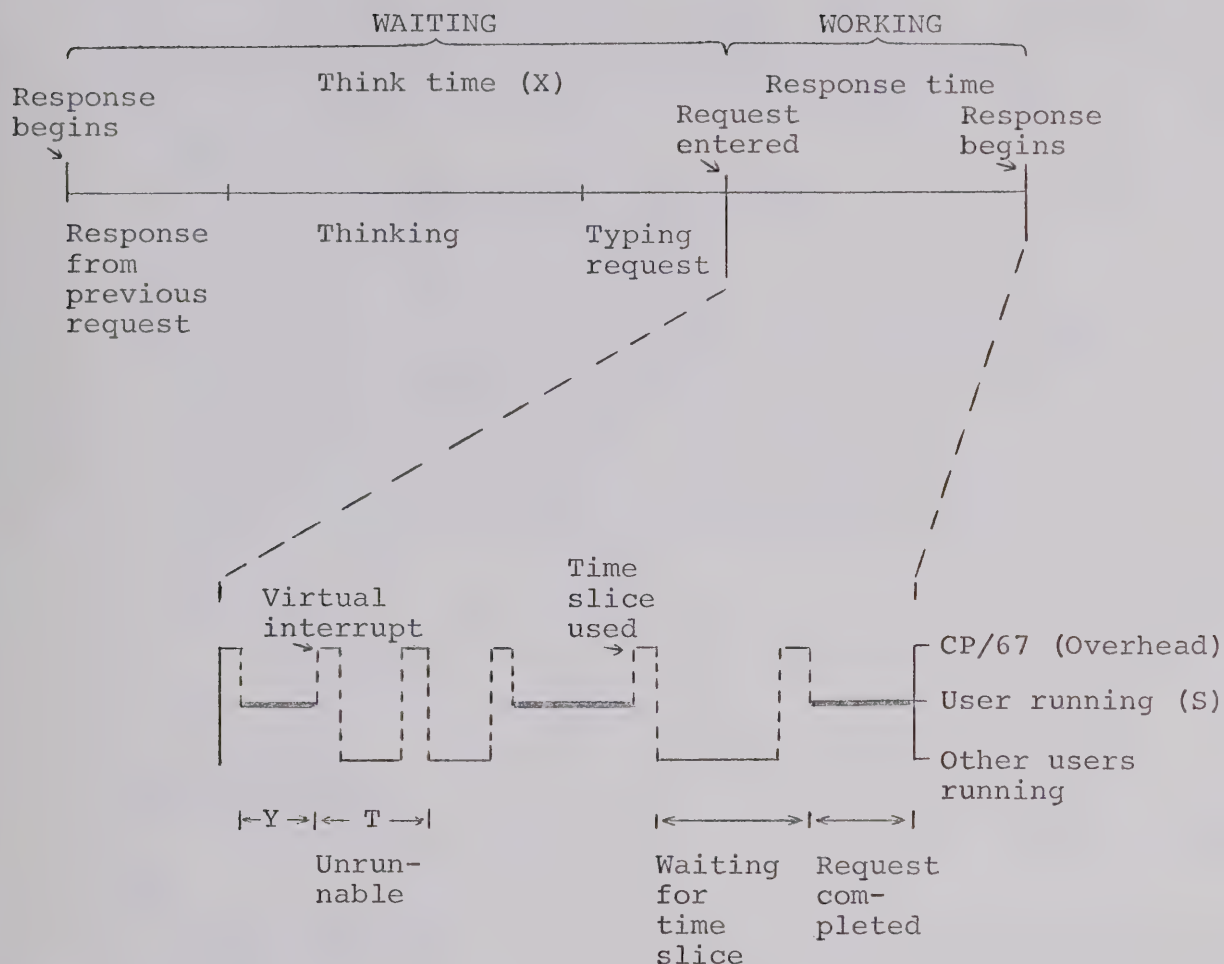


Fig. 5.1 User Model

executing the request, the response time includes such times as that spent waiting to enter a queue, waiting for a time-slice (while some other request is running), waiting in an unrunnable state (e.g. PAGEWAIT or IOWAIT), and system overhead.

The input demands on a system can be represented mathematically in a model by the following two descriptions:

a) the frequency of the demands

b) the size of the demands.

In our model, the individual user is described by two such pairs of demand descriptions. The first pair describe the user's request, while the second pair describe the virtual interrupt requests (see Fig. 5.2).

User described

by

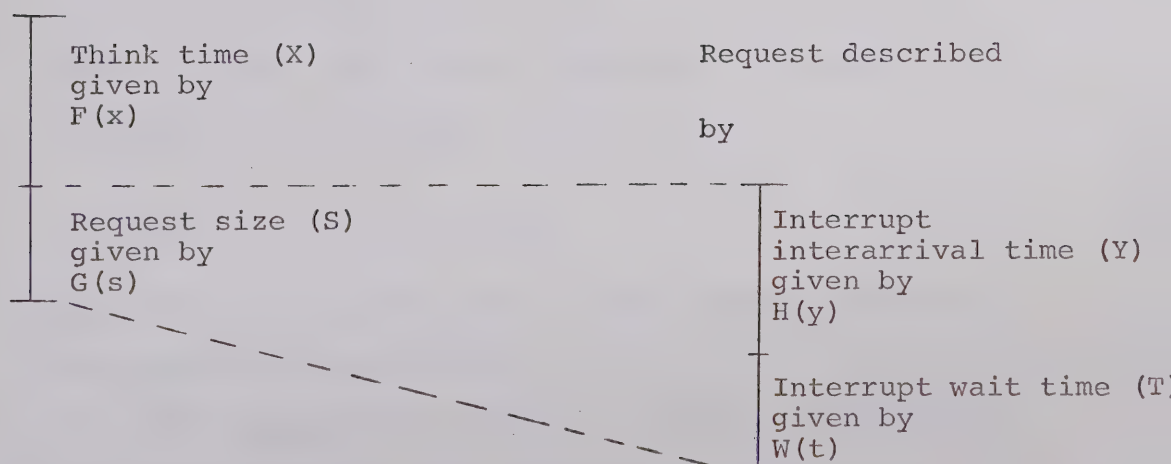


Fig. 5.2 Description of Model Input

The virtual interrupt request could be considered a second level description of the user. On the first level, a user is described by the size of his requests and their rate of arrival. Each request, in turn, is described by the size of its virtual interrupts and their rate of arrival. (Higher levels of input description could be reached by further describing each interrupt as to its type or cause; however this would require a much greater degree of analysis of the real system than was feasible in this study.)

More specifically, the frequency of user requests for processor time is obtained from the think time distribution

$$F(x) = \text{Prob } (X \leq x) ,$$

where X is the think time. The size of requests is obtained from the request size distribution

$$G(s) = \text{Prob } (S \leq s) ,$$

where S is the request size. These distributions are independent of each other.

The frequency of virtual interrupts is obtained from the interrupt interarrival time distribution

$$H(z) = \text{Prob } (Y \leq z \mid S') ,$$

where Y is the amount of processing time between the occurrence of virtual interrupts within a request and S' is the length of time that a request has already executed ($S' < S$) (see Chapter VI, Sec. 1.1.3). The size of the virtual interrupts is obtained from the interrupt wait time distribution

$$W(t) = \text{Prob } (T \leq t) ,$$

where T is the length of time a user is unrunnable while CP/67 is processing this interrupt.

All distributions used to describe the user's input, except $W(t)$, were obtained directly from the measurement data (see Chapter VI).

Given this model of a user, the behavior of CP/67 will be simulated by modeling the interactive activities of several of these users as controlled by DISPATCH.

1.2 The GPSS DISPATCH Model

Several general purpose simulation languages have been developed recently as the use of simulation has grown. Such languages are designed to relieve the analyst of much detailed programming and speed up the process of obtaining results. Teichroew and Lubin [37]

give an excellent survey of available simulation languages and the considerations which enter into the selection of any such language. In this project GPSS (General Purpose Simulation System) was selected because of its availability.

The language GPSS [17,18] is based on the assumption that any system can be represented in terms of a small set of abstract elements called "entities". Likewise the logical rules governing any system can be reduced to a common set of simple operations. There are four basic classes of entities around which GPSS operates: (1) dynamic, (2) equipment, (3) operational, and (4) statistical.

The dynamic entities in GPSS are called "transactions". They are used to represent elements of the system which are thought of as moving through the system. Associated with each transaction is a set of parameters which can be assigned values to represent the characteristics of that transaction. In our model of DISPATCH, the CP/67 users become transactions which move through the operational entities (see below) that model the logical operation of DISPATCH. The parameters of each such transaction are used to keep information on the status of the corresponding user (e.g. USERID, runnable, unrunnable, in Q_1 , in Q_1 -WAIT, etc.). The movement of the user transaction through

the model of DISPATCH can be compared with the movement of the user's UTABLE through the real DISPATCH of CP/67.

The equipment entities in GPSS represent elements of the system's equipment that are acted upon by the transactions. In our model the CPU, for example, becomes a "facility" which can be "seized" by only one transaction at a time. Thus the user transaction that is chosen to run by DISPATCH will seize the CPU. The length of time the user has control of the CPU represents the time his request requires for execution in the real system. Besides the user transactions, there are two other types of transactions which can control the CPU. A "DISPATCH" transaction has control during the time represented by the execution of DISPATCH. An "interrupt" transaction has control during the handling of each interrupt. The only time the CPU is not under the control of a transaction is when the idle state is entered.

The operational entities of GPSS are called "blocks" and constitute the logic of the system, instructing the transaction where to go and what to do next. There are 43 distinct block types in GPSS, each representing some step in the operation of the

system. The functioning of GPSS is based on the entry of transactions into blocks and the subsequent execution of the GPSS subroutine associated with each block type. Four basic types of events may occur within a block:

- a) creation or destruction of a transaction;
- b) alteration of a numeric attribute of some entity;
- c) delay of a transaction for some specific amount of time; or
- d) alteration of the transaction flow through the block network.

For example, in our model the occurrence of a virtual interrupt during execution results in:

- a) the creation of an interrupt transaction;
- b) the user transaction is made unrunnable;
- c) the interrupt transaction is delayed for the assigned interrupt wait time (T), during which the user remains unrunnable;
- d) when time T has passed the interrupt transaction preempts control of the CPU, terminating the current request transaction. (The user is made runnable again, the interrupt transaction is destroyed, and control returns to DISPATCH.)

The final class of entities built into GPSS is the statistical entities which provide the facilities for obtaining measurements on the behavior or performance of the simulated system. For example, GPSS "tables" were used extensively in this project to obtain the frequency distributions of several events occurring in the model. "Queues" were used to obtain measurements of the activity at the user queues maintained by DISPATCH.

Each movement of a transaction from block to block is considered by GPSS as an event which is scheduled to occur at some particular time. In order to provide the correct time sequence, GPSS maintains a clock of real simulated time. At each point in time when some event is scheduled to occur, GPSS will process all currently active transactions that can be moved into some next block. The clock is then updated to the time of the next scheduled event and, in this fashion, the passage of time is simulated.

The time unit represented by the clock is selected by the GPSS user, depending on the objectives of the project and, to some extent, on the accuracy of the input data. In this model, the unit chosen was one msec. It was felt that a unit of this size was necessary to model the operation of the system in sufficient detail,

even though some of the input data was based on samples taken every 100 msec. (or .1 sec.).

Fig. 5.3 illustrates the GPSS logic used to simulate the flow of a user's requests through CP/67. (For further information on the input requirements and program parameters see Appendix C, and for program listing contact the Department of Computing Science Library at the U of A.) The logic used to simulate the operation of DISPATCH is, of course, the same as that of the real system as described in Chapter IV. The following simplifications were made however.

- a) There was no distinction made between types of interrupts. For example, the user was not put in IOWAIT or PAGEWAIT; rather, he was made either runnable or unrunnable. Although the crucial reason for not simulating the types of interrupts was the lack of accurate data, it was felt that such detail was not necessary within the objectives of the project. The important consideration was accurate representation of the virtual interrupts and their influence on the operation of DISPATCH.
- b) As a direct result of the first simplification, the operation of DISPATCH did not include the examination for CPRQUEST, or for PENDING interrupts.

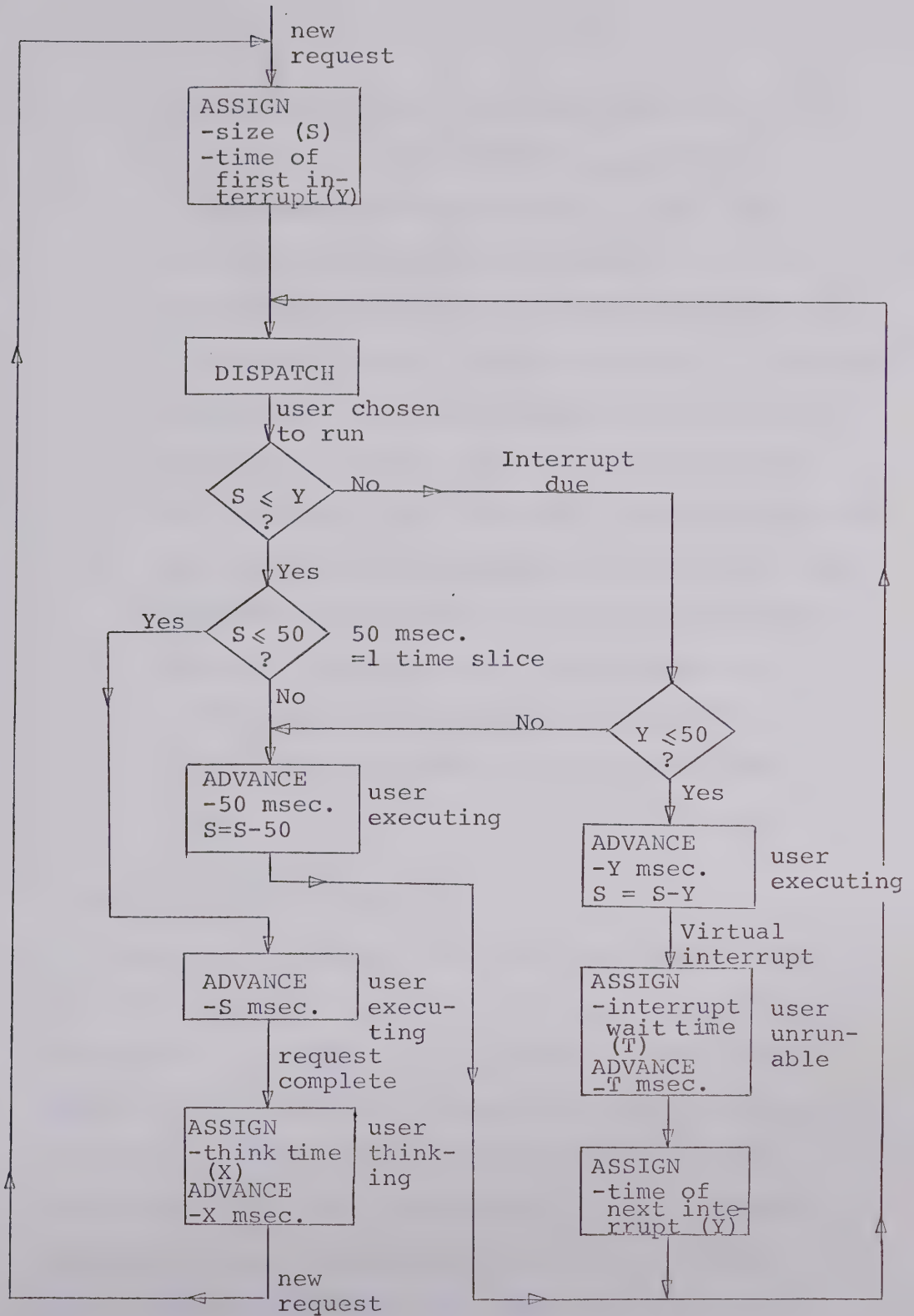


Fig. 5.3 GPSS User Model

c) In the real system, a user in execution is put in CFWAIT each time a response is required from the terminal. A user found in CFWAIT is considered interactive and is put in Q_1 -WAIT (giving him the fastest possible attention when the terminal response is received). Since the signal to terminate the CFWAIT condition is from the terminal hardware itself, not from the user, and since the data collected was not able to distinguish between these two signals, the requests modeled likewise do not make this distinction. A new request is defined each time a user enters Q_1 -WAIT, regardless of whether it is entered from CFWAIT or WAIT (i.e. an actual user request).

2 The Measurement Technique

As stated earlier, the technique used to collect data from the CP/67 system is an extension of the method developed by A. Gatha [15] in an earlier evaluation study of the U of A installation. The method involves using a CMS virtual machine which is able to read information from the CP/67 nucleus. It is a sampling technique in which the relevant data is obtained at certain short time intervals. The method adds no

measurable overhead to the operation of CP/67 since the data gathering program is just another normal CMS program. Its operation is designed to take advantage of the time-sharing nature of the system to provide the repeated bursts of execution necessary to obtain the proper sampling.

There are two qualifications to make on the term "normal CMS program" used above, since the virtual machine used (CPSTATS) is privileged in two respects. First, it was given a privileged priority class which enables CPSTATS to execute the "diagnose" privileged instruction (DIAG). (All these programs are in 360/Assembler language.) The DIAG instruction allows the user to read values from specified locations within the CP/67 supervisor.

Second, CPSTATS is privileged by the addition of a "real" timer. This is an important extension of the basic method used by Gatha, since it allowed the author to develop a program which would collect data samples at considerably smaller time intervals. The normal CMS user receives a timer interrupt after every 2 sec. of execution time. This timer interrupt is virtual; that is, the timer interrupt is simulated by CP/67 each time it finds that a user has used an additional 2 sec. of VTOTTIME. The virtual machine

does not "receive" the interrupt until it is chosen to run again. The "real" timer in CPSTATS is also virtual; however the time used by CP/67 to calculate the passage of time between interrupts consists, not only of CPSTATS's VTOTTIME, but also the time spent by CPSTATS in WAIT state. Thus it does in fact represent all real time for that virtual machine. CPSTATS must still wait its turn before it can process the interrupt. (These two privileged conditions exist within CP/67 and can be implemented by specifying, at system generation time, that the virtual machine has these capabilities.)

In the next section the basic logic of the data gathering package is discussed. Following this the specific data collected will be described.

2.1 Program Logic

The first important consideration in developing the program logic was how to control the time intervals at which data samples would be taken. An interval of .1 sec. was deemed necessary to provide sufficient information for describing the user being "followed". Implementation of the "real" timer gave the control that was needed. In a normal CMS virtual machine a timer interrupt merely produces a "blip" at the user's

terminal, and does not otherwise affect the execution of the user's present request. In CPSTATS, the data gathering routines are inserted into the CMS external interrupt handler (EXTINT), and each time CPSTATS receives a timer interrupt these routines are executed. The interrupt interval (in EXTINT) is altered from the normal 2 sec. to provide an interrupt approximately every .1 sec. real time.

Initiating each data collection run involves specifying the USERID of the user to be followed that day (OURMAN), and altering CPSTATS's EXTINT routine to include branch to the actual data gathering programs as well as to insert the new interrupt time interval. Once EXTINT has been altered, the collection of data automatically begins with the first timer interrupt. Thus, during a data collection run, the CPSTATS virtual machine is not doing any work other than receiving timer interrupts and executing the altered version of EXTINT.

There are two routines associated with the collection of data. The first (WAITEST) receives control from EXTINT and performs a preliminary examination of OURMAN to determine if his status has changed since the last search and, if not, whether the last data record should be saved. The second routine (UDATA) is then given control and performs the actual data

collection. Control is then returned to EXTINT and CPSTATS enters the WAIT state to await the next timer interrupt.

WAITEST

Two considerations were involved in the development of WAITEST.

- a) With data being collected every .1 sec. on the average, the accumulated volume of data becomes very large and provisions had to be made to store it.
- b) Any data taken while OURMAN is inactive (i.e. thinking) yields no useful information, since he is representing no new input to the system.

WAITEST is thus designed to save only that data which represents the user's periods of activity. The data collected is in a condensed form eliminating the excess bulk of repetitive records, and allowing a longer collection period.

WAITEST operates in the following manner.

- a) The address of OURMAN's UTABLE is found by searching down the NEXTUSER chain of UTABLEs, using the DIAG instruction. (At this stage WAITEST will also detect if OURMAN has signed off, and, if so, the data gathering run is terminated.)

b) When OURMAN has been found, WAITEST gathers certain information from his UTABLE and performs an initial examination.

- i) If OURMAN is in WAIT, and has been since the last examination, the output buffer pointer used by UDATA (see below) is moved back so that the data record obtained in the previous search will be overwritten.
- ii) If OURMAN is not in WAIT, or if he has executed since the last examination, the buffer pointer is not moved back and control is transferred directly to UDATA. This time the data from the previous search is saved and the current data is placed in the next buffer location.

The effect of WAITEST's operation is that the data collected produces a series of "snapshot" descriptions of OURMAN. Each time OURMAN is active, data samples are obtained with each interrupt. When he is inactive for more than two searches there is a corresponding gap in the data. A count is maintained of the number of searches "lost" during each inactive period.

UDATA

The function of UDATA is simply to collect the specified data each time it is entered. The major consideration is the storing of the data. Each time it executes, UDATA withdraws 64 bytes of raw data from the CP/67 nucleus. UDATA provides an 800 byte buffer area within itself for temporary data storage. As data is put into the buffer, a pointer is advanced so that information is not overwritten, unless WAITEST deliberately moves the pointer back. After the data is collected, control is returned to EXTINT.

Two conditions may alter UDATA's basic operation.

- a) If the pointer moves beyond the 800 byte buffer limit, meaning the buffer is full, the data in the buffer is transferred to intermediate disk storage.
- b) If, when the buffer is read out onto disk, the intermediate storage area also becomes full (see below), then the data gathering run is terminated.

The size of the CPSTATS virtual machine is defined to be 896 records of 800 bytes each. Since some of this space is required to contain the data gathering package

itself, a limit of 800 records was set aside for intermediate storage of data. When this space is full, the data gathering run must terminate.

A data gathering run is terminated in one of three ways:

- a) WAITEST automatically terminates when it finds that OURMAN has signed off;
- b) UDATA automatically terminates when intermediate storage is filled; and
- c) the CPSTATS user can terminate the run at any time by issuing the CMS "kill execution" (kx) command at the terminal.

When a run is terminated, another program (ANUSER) is executed which takes the raw data, which is in binary form, from intermediate disk storage, edits it for the desired information, converts this to decimal, and then punches it out on cards. For each recorded search, one card is punched. When the data has been transferred to permanent storage (i.e. cards), the intermediate storage can be cleared for the next run.

The amount of data produced during one run will depend on the length of that run and the activity of OURMAN. Since a user who is active 100% of the time will produce 64 bytes (and 1 card) of data each .1 sec., one buffer (800 bytes) will hold the

results of 12 searches or 1.2 sec., and intermediate storage will hold 960 sec. (16 min.) of data. Hence the data gathering could be as short as 16 min. and could produce as many as 9600 cards. The average, however, was about one hour, with about 5500 cards being produced.

2.2 The Data Collected

Each search of UDATA produces a complete set of the variables listed below. The significance of each variable is given where it is felt necessary. Many have already been introduced. Most of the variables used as data already existed within the CP/67 system; however a few had to be added especially for this project. Such variables are marked with an (*).

The following variables were obtained from OURMAN's UTABLE.

- a) VMSTATUS indicates whether or not OURMAN is in either PAGEWAIT, IOWAIT, or CFWAIT.
- b) TIMINQ indicates whether or not OURMAN is in a queue, and, if so, in which queue. If he is in a queue, TIMINQ contains his TIMEUSED at the time he entered that queue. By comparing this value with his present TIMEUSED, it can be determined how long he has been in that queue. Also, TIMINQ contains the NOQUANT value of OURMAN.

- c) VPSW indicates whether or not OURMAN's virtual machine is in WAIT state.
- d) VTOTTIME indicates how much time the virtual machine has spent in actual execution. The size of a request can be determined by examining the before-and-after values of VTOTTIME.
- e) USERINTD (*) indicates how many times OURMAN has been interrupted while executing.
- f) USERGO (*) indicates how many times OURMAN was interrupted while executing but was chosen to run again immediately (Choice 1; see Chapter IV, Sec. 4.2.1.2). The difference between USERINTD and USERGO gives us the number of times OURMAN could not run again immediately because he was unrunnable. Hence we know how many virtual interrupts he has caused.
- g) INTPTIME (*) indicates the amount of TIMEUSED which has been charged to OURMAN, by CP/67, for handling his virtual interrupts.
- h) NUMPAGES indicates the number of pages OURMAN has in core at the present time. (This value was taken for interest sake and was not used in the simulation study.)

The following variables are system parameters obtained from fixed locations in the CP/67 nucleus.

- i) HOURS indicates the real time of day in hours, minutes, and seconds. It is used to measure time intervals which exceed one second in length. For time intervals in the neighbourhood of one second or less, the length of the interval is measured by the number of searches involved, assuming each search is equal to .1 sec.
- j) TIMEUSED.
- k) OVERHEAD.
- l) CPTIME.
- m) WAITTIME.
- n) DISPENT (*) indicates the number of times DISPATCH was entered after finding and executing a CPRQUEST.
- o) DISPENT+4 (*) indicates the number of times DISPATCH was entered normally - after a virtual machine had been executing.
- p) DISPENT+8 (*) indicates the number of times DISPATCH was entered from WAIT state. The sum of the last three entries yields the total number of times DISPATCH was entered.

In addition:

q) WAITCNT indicates the number of searches "lost" since the last active period of OURMAN. This contains a value only for the first search of a new request. It is used to calculate the length of the preceding think time where this interval is less than one second. Otherwise HOURS is used.

This concludes the description of the simulation model and the measurement technique used in this project. The next chapter discusses the results obtained from the data gathered and from the operation of the simulation model.

CHAPTER VI

RESULTS

Measurements were taken over a 6 week period of time during March and April, 1970. During that time CP/67 ran $3\frac{1}{2}$ hours a day, except Sunday, between the hours of 8:15 and 10:45 a.m.. Data was taken only on weekdays and generally after 9 a.m., when a "normal" load was considered to exist on the system. To determine what constituted a normal load, a rough survey was taken over the duration of the data collection period. The survey showed a minimum of 5 users on the system and a maximum of 16 users, with the average being about 10 at the beginning of the tests and increasing to approximately 13 near the end of the tests.

On any time-sharing system it is impossible to define a normal load, since users are continually signing on and off. Also each user may represent an entirely different load to the system. For example, the figures given above include such users as CPSTATS, APL, and OPER (the operator) which would probably be considered unusual in any definition of normal.

A total of 14 users were followed for the purpose of obtaining representative data for input to the simulation model. The users chosen were picked arbitrarily

from among those signed on. Although there were close to 100 registered CMS users at the U of A, only 20-30 were active to any extent, and efforts were made to follow users who were recognized as common users of CP/67.

A follow-up survey was taken of each user on the day he was observed. Results showed that these users were signed on for an average of about 1 hr. and 20 min. During their terminal session they used an average of about 1 min. 10 sec. execution time (VTOTTIME) but were charged for about 2 min. 15 sec. (TIMEUSED). The ratio of TIMEUSED to VTOTTIME was about 2:1. This ratio appeared to be directly proportional to the amount of I/O conducted during the terminal session. It ranged from 1.1:1 for a compute-bound user, to 4.7:1 for an I/O bound user. (The more interrupts a user caused, the more work that must be done by CP/67 to handle them, and hence the I/O user was charged more.)

In addition to the 14 (normal CP/67) users followed for data collection purposes, a run was made on each of APL and CPSTATS. During the data collection period, APL was used on a limited basis in conjunction with CP/67. Although APL is considered a single virtual machine by CP/67, it is, in fact, a time-sharing system in itself which can handle many users on this single

machine. Although it only had 3 users signed on, results of the APL run showed it to be a very heavy user of computer time. During the 11 min. 13 sec. of data collection, APL used 7 min. 26 sec. of VTOTTIME, or 66% of the CPU time available! (It is thought that the APL users were probably taking advantage of the low usage of APL on CP/67 to execute large programs resulting in the heavy computing load.) Because of the unusual nature of the APL user, and the heavy load it represented, the data obtained from it was not considered as representing normal CP/67 user input and therefore it was not used.

A run was made on CPSTATS itself to determine how much load the data gathering machine was imposing on the system. During the 16 min. 14 sec. (774 sec.) run, CPSTATS was charged with 23 sec. of TIMEUSED, of which 5 sec. was VTOTTIME - a ratio of 4.6:1. The 23 sec. represented 2.3% of the CPU time available during the period. The 5 sec. VTOTTIME alone represented only $\frac{1}{2}\%$. The high TIMEUSED to VTOTTIME ratio indicated that a good deal of the load from CPSTATS is I/O, which was the movement of data into intermediate disk storage. Since there were 10,561 searches made, each one involved an average of about 2.2 msec. of TIMEUSED. CPSTATS occupied between 5 to 6 pages in core. This was about 3% of a total of 192 pages of core storage available at the U of A.

1 Measurement Results

A Fortran program was written which analyzed the data collected. The results of each sample were compiled into cumulative tables using APL. The results of the analysis are presented in this section.

The 14 users were followed for an average of 54 min. each. This involved a total of 457,331 searches for an average of one search every .10 sec. as desired. Among individual samples this figure varied from .07 sec. to .16 sec.

1.1 User Description

1.1.1 Request Size Distribution

A total of 12465 user requests were recorded. Table 6-1 classifies these requests by user and by size. The intervals used to classify the requests by size were based on the operation of the real clock in the 360/67 which increments by one every 3.3 msec. Hence the smallest measurable time increment was approximately 3 msec. It was found that 5577, or 44.7%, of all requests were under 3 msec. in size, 3598, or 28.9%, were between 3 and 6 msec., and so on (see Table 6-1).

Table 6-2 gives the cumulative distribution of request sizes for all users as taken from Table 6-1. From this a classification of interactive and heavy

TABLE 6-1 REQUEST SIZES

Req. size (msec)	User no.														Total
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
151	1726		431	269	165	308	106	899	397	286	86	72	514	167	5577
32	552		570	187	90	201	169	151	358	823	47	69	266	83	3598
1	476		77	71	15	45	4	4	64	275	21	29	53	8	1143
12	24		16	23	6	24	0	1	51	58	8	17	18	0	247
15	0		3	18	6	18	0	0	33	5	3	1	26	0	116
20	6		7	36	5	24	0	7	49	21	4	6	20	0	187
25	2		3	15	0	7	0	11	13	2	2	2	9	0	67
30	8		13	23	10	21	0	50	23	11	0	1	5	0	180
35	41		13	4	3	6	0	36	8	8	1	2	7	0	132
40	236		9	16	4	6	1	27	7	5	1	0	7	0	322
45	4		3	58	7	1	1	6	0	1	0	2	2	0	31
50	1		5	4	1	5	1	0	1	4	0	1	1	0	80
100	40		10	73	16	49	11	3	8	21	0	2	10	0	244
200	2		43	24	12	36	0	0	9	20	16	3	15	1	181
300	0		1	7	2	9	2	0	3	21	0	4	10	0	59
400	0		1	4	0	14	5	0	0	12	2	2	5	0	46
500	0		0	4	1	6	0	0	2	0	0	1	2	0	16
600	0		1	2	3	6	0	0	0	0	0	1	0	1	14
700	1		0	0	1	0	0	0	1	1	0	79	2	0	86
800	1		0	1	3	0	0	0	2	0	0	2	2	0	11
900	0		0	0	0	1	0	0	1	1	2	0	0	0	5
1000	0		1	0	0	2	0	0	0	0	0	0	0	0	3
2000	0		0	3	2	5	0	0	5	8	25	19	0	0	71
3000	0		0	1	0	4	0	0	0	1	9	0	0	0	15
4000	0		0	1	0	1	0	0	0	4	3	0	0	0	9
5000	0		0	0	0	0	0	0	0	2	7	0	0	0	9
over 5000	0		0	3	0	1	0	0	0	5	7	0	0	0	16
Total	218	3123	1207	847	352	800	300	1195	1035	1595	244	315	974	260	12465

TABLE 6-2 REQUEST SIZE DISTRIBUTION (a)

Req. size (msec)	User no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Overall
3	69.3	55.3	35.7	31.8	46.9	38.5	35.3	75.2	38.4	17.9	35.2	22.9	64.2	52.8	64.2	44.7
6	83.9	72.9	82.9	53.8	72.4	63.6	91.7	87.9	72.9	69.5	54.5	44.8	96.2	80.1	96.2	73.6
9	84.4	88.2	89.3	62.2	76.7	69.2	93.0	88.2	79.1	86.8	63.1	54.0	99.2	85.5	99.2	82.8
12	84.9	89.0	90.6	64.9	78.4	72.2	93.0	88.3	84.1	90.4	66.4	59.4	99.2	87.4	99.2	84.8
15	86.2	89.0	90.9	67.1	80.1	74.5	93.0	88.3	87.2	90.7	67.6	59.7	99.2	90.0	99.2	85.7
20	87.2	89.1	91.5	71.3	81.5	77.5	93.0	88.9	92.0	92.0	69.3	61.6	99.2	92.1	99.2	87.2
25	87.6	89.2	91.7	73.1	81.5	78.4	93.0	89.8	93.2	92.2	70.1	62.2	99.2	93.0	99.2	87.7
30	94.5	89.5	92.8	75.8	84.4	81.0	93.0	94.0	95.5	92.9	70.1	62.5	99.2	93.5	99.2	89.2
35	95.9	90.8	93.9	76.3	85.2	81.7	93.0	97.0	96.2	93.4	70.5	63.2	99.2	94.3	99.2	90.2
40	97.2	98.3	94.6	78.2	86.4	82.5	93.3	99.2	96.9	93.7	70.9	63.2	99.2	95.0	99.2	92.8
45	97.2	98.5	94.9	78.6	88.4	82.6	93.7	99.7	96.9	93.7	70.9	63.8	99.2	95.2	99.2	93.1
50	98.2	98.5	95.3	85.5	88.6	83.2	94.0	99.7	97.0	94.0	70.9	64.1	99.2	95.3	99.2	93.7
100	98.6	99.8	96.1	94.1	93.2	89.4	97.7	100.0	97.8	95.3	70.9	64.8	99.2	96.3	99.2	95.7
200	98.6	99.8	99.7	96.9	96.6	93.9	97.7	100.0	98.6	96.6	77.5	65.7	99.6	97.8	99.6	97.1
300	98.6	99.8	99.8	97.8	97.2	95.0	98.3	100.0	98.9	97.9	77.5	67.0	99.6	98.9	99.6	97.6
400	99.1	99.8	99.8	98.2	97.2	96.7	100.0	100.0	98.9	98.6	78.3	67.6	99.6	99.4	99.6	98.0
500	99.1	99.8	99.8	98.7	97.4	97.5	100.0	100.0	99.1	98.6	78.3	67.9	99.6	99.6	99.6	98.1
600	99.1	99.8	99.9	98.9	98.3	98.2	100.0	100.0	99.1	98.6	78.3	68.3	100.0	99.6	100.0	98.2
700	99.5	99.9	99.9	98.9	98.6	98.2	100.0	100.0	99.2	98.7	78.3	93.3	100.0	99.8	100.0	98.9
800	100.0	99.9	99.9	99.1	99.4	98.2	100.0	100.0	99.4	98.7	78.3	94.0	100.0	100.0	100.0	99.0
900	100.0	99.9	99.9	99.1	99.4	98.4	100.0	100.0	99.5	98.7	79.1	94.0	100.0	100.0	100.0	99.0
1000	100.0	99.9	100.0	99.1	99.4	98.6	100.0	100.0	99.5	98.7	79.1	94.0	100.0	100.0	100.0	99.0
2000	100.0	100.0	100.0	99.4	100.0	99.2	100.0	100.0	100.0	99.2	89.3	100.0	100.0	100.0	100.0	99.6
3000	100.0	100.0	100.0	99.5	100.0	99.7	100.0	100.0	100.0	99.3	93.0	100.0	100.0	100.0	100.0	99.7
4000	100.0	100.0	100.0	99.6	100.0	99.9	100.0	100.0	100.0	99.6	94.3	100.0	100.0	100.0	100.0	99.8
5000	100.0	100.0	100.0	99.6	100.0	99.9	100.0	100.0	100.0	99.7	97.1	100.0	100.0	100.0	100.0	99.9
over 5000	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

TABLE 6-3 REQUEST SIZE DISTRIBUTION (b)

Req. size (msec)	NORMAL			INTERACTIVE			HEAVY		
	No. req.	Per cent.	Cum. dist.	No. req.	Per cent.	Cum. dist.	No. req.	Per cent.	Cum. dist.
3	5577	44.7	44.7	4129	47.7	47.7	158	28.3	28.3
6	3598	28.9	73.6	2614	30.2	77.9	116	20.8	49.0
9	1145	9.2	82.8	897	10.4	88.3	50	8.9	58.0
12	247	2.0	84.7	117	1.4	89.6	25	4.5	62.4
15	116	0.9	85.7	54	0.4	90.0	4	0.7	63.1
20	187	1.5	87.2	61	0.7	90.7	10	1.8	64.9
25	67	0.5	87.7	27	0.3	91.0	4	0.7	65.6
30	180	1.4	89.2	87	1.0	92.0	1	0.2	65.8
35	132	1.1	90.2	105	1.2	93.3	3	0.5	66.4
40	322	2.6	92.8	285	3.3	96.6	1	0.2	66.5
45	31	0.3	93.0	17	0.2	96.8	2	0.4	66.9
50	80	0.6	93.7	12	0.1	96.9	1	0.2	67.1
100	244	2.0	95.6	95	1.1	98.0	2	0.4	67.4
200	181	1.5	97.1	81	0.9	98.9	19	3.4	70.8
300	59	0.5	97.6	34	0.4	99.3	4	0.7	71.6
400	46	0.4	97.9	23	0.3	99.6	4	0.7	72.3
500	16	0.1	98.1	2	0.0	99.6	1	0.2	72.4
600	14	0.1	98.2	2	0.0	99.6	1	0.2	72.6
700	86	0.7	98.9	4	0.0	99.7	79	14.1	86.8
800	11	0.1	99.0	2	0.0	99.7	2	0.4	87.1
900	5	0.0	99.0	1	0.0	99.7	2	0.4	87.5
1000	3	0.0	99.0	1	0.0	99.7	0	0.0	87.5
2000	71	0.6	99.6	12	0.1	99.9	44	7.9	95.3
3000	15	0.1	99.7	1	0.0	99.9	9	1.6	97.0
4000	9	0.1	99.8	4	0.0	99.9	3	0.5	97.5
5000	9	0.1	99.8	2	0.0	99.9	7	1.3	98.7
over 5000	16	0.1	100.0	5	0.1	100.0	7	1.3	100.0
Total	12465			3654			559		

(compute-bound) users was made. Interactive users were defined to be those users for which 85% of their requests were less than 9 msec. This included 7, or 50%, of the observed users (Users no. 2, 3, 7, 8, 10, 13, and 14). Heavy users were defined as those with approximately 30% of their requests over 50 msec. This included 2 of the users, namely no. 11 and 12.

Table 6-3 is obtained from Table 6-2 by averaging the interactive and heavy users and thus shows the three request size distributions as they were inserted into the simulation model. A normal user is defined by the distribution obtained over all 14 samples. The maximum request size put into the simulation model was 12 sec. The 16 overflow requests (over 5 sec. in size) recorded averaged a little over 8 sec. each. The highest value found was given by user no. 4 whose 3 overflow requests averaged 14.5 sec. each.

Although 82% of all requests were less than 9 msec., the average over all requests measured was 46 msec. (The midpoint of each interval was used in this calculation.)

1.1.2 Think Time Distribution

Table 6-4 gives the frequency distribution of the think time values found for the (12465) requests recorded.

TABLE 6-4 THINK TIME DISTRIBUTION (a)

Think time (sec)<	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Total
0	65	855	885	405	62	244	13	297	601	465	122	89	244	13	4365
1	30	1292	170	124	41	127	31	224	125	199	35	39	188	4	2629
2	20	588	49	100	50	128	37	138	110	137	12	27	191	3	1590
3	7	171	9	63	58	82	19	91	27	336	17	31	108	3	1022
4	7	70	16	34	20	41	13	100	26	89	4	25	58	7	510
5	44	47	11	30	14	25	9	239	13	195	3	13	45	22	710
6	32	33	6	8	13	24	9	103	16	43	11	20	30	150	498
7	1	16	2	8	4	21	4	2	18	15	5	46	12	41	195
8	0	8	2	9	2	17	10	0	16	8	3	15	10	2	102
9	0	11	3	7	4	8	13	0	19	7	1	0	12	1	86
10	0	6	2	9	2	9	8	0	14	46	3	0	6	1	106
15	1	13	2	23	13	39	50	0	36	42	10	3	19	3	254
20	5	4	5	10	13	13	31	0	9	5	7	1	13	0	116
25	0	2	6	8	8	9	24	0	3	1	4	2	9	1	77
30	1	2	7	5	9	5	17	0	2	1	3	1	7	2	62
35	0	0	8	0	5	7	2	0	0	2	1	0	1	1	27
40	0	1	5	0	4	0	7	0	0	1	2	1	2	0	23
45	1	1	3	3	4	1	2	0	0	1	0	0	2	0	21
50	1	1	3	0	4	1	0	0	0	0	0	0	0	0	9
55	1	0	1	1	3	0	0	0	0	1	0	1	2	1	11
60	1	0	1	0	1	0	1	0	0	0	0	0	2	0	6
120	1	1	2	0	18	0	0	0	0	1	1	1	5	0	29
180	1	1	1	0	0	0	0	0	0	1	1	0	6	0	11
240	0	0	1	0	0	0	0	0	0	0	0	0	1	0	3
300	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
over 300	0	0	1	0	0	0	0	1	0	0	0	0	1	0	3

TABLE 6-5 THINK TIME DISTRIBUTION (b)

Think time (sec)<	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Total
1	61	391	934	419	57	244	4	174	620	457	131	87	230	3	3812
2	8	157	12	5	0	11	0	79	9	70	2	4	5	7	369
3	10	140	11	16	1	16	0	83	17	40	2	5	17	1	359
4	10	181	12	20	1	16	0	53	13	38	1	1	14	8	368
5	2	213	15	9	2	9	2	37	6	15	4	2	18	2	336
6	2	250	11	7	4	5	3	19	12	6	3	3	21	0	346
7	0	222	6	11	5	4	1	10	10	5	2	1	12	0	289
8	0	75	14	8	3	5	3	10	15	2	2	2	8	0	147
9	1	46	5	7	2	3	5	8	9	7	4	0	15	1	113
10	1	472	35	27	28	58	26	48	15	24	6	23	92	0	855

TABLE 6-6 THINK TIME DISTRIBUTION (c)

Think time (sec)<	NORMAL			INTERACTIVE			HEAVY		
	Freq.	Per cent.	Cum. dist.	Freq.	Per cent.	Cum. dist.	Freq.	Per cent.	Cum. dist.
1	3812	30.6	30.6	2193	25.3	25.3	218	39.0	39.0
2	369	3.0	33.5	330	3.8	29.2	6	1.1	40.1
3	359	2.9	36.4	292	3.4	32.5	7	1.3	41.3
4	368	3.0	39.4	306	3.5	36.1	2	0.4	41.7
5	336	2.7	42.1	302	3.5	39.6	6	1.1	42.8
6	346	2.8	44.8	310	3.6	43.1	6	1.1	43.8
7	289	2.3	47.2	256	3.0	46.1	3	0.5	44.4
8	147	1.2	48.3	112	1.3	47.4	4	0.7	45.1
9	113	0.9	49.2	87	1.0	48.4	4	0.7	45.8
10	855	6.9	56.1	697	8.1	56.4	29	5.2	51.0
11	1590	12.8	68.9	1143	13.2	69.7	39	7.0	58.0
12	1022	8.2	77.1	737	8.5	78.2	48	8.6	66.5
13	510	4.1	81.2	353	4.1	82.3	29	5.2	71.6
14	710	5.7	86.9	568	6.6	88.8	16	2.9	74.5
15	498	4.0	90.8	374	4.3	93.1	31	5.5	80.1
16	195	1.6	92.4	92	1.1	94.2	51	9.1	89.3
17	102	0.8	93.2	40	0.5	94.7	18	3.2	92.5
18	86	0.7	93.9	47	0.5	95.2	1	0.2	92.7
19	106	0.9	94.8	69	0.8	96.0	3	0.5	93.2
20	254	2.0	96.8	129	1.5	97.5	13	2.3	95.5
21	116	0.9	97.7	58	0.7	98.2	8	1.4	97.0
22	77	0.6	98.4	43	0.5	98.7	6	1.1	98.0
23	62	0.5	98.9	36	0.4	99.1	4	0.7	98.7
24	27	0.2	99.1	14	0.2	99.2	1	0.2	98.9
25	23	0.2	99.3	16	0.2	99.4	3	0.5	99.5
26	21	0.2	99.4	13	0.2	99.6	0	0.0	99.5
27	9	0.1	99.5	4	0.0	99.6	0	0.0	99.5
28	11	0.1	99.6	5	0.1	99.7	1	0.2	99.6
29	6	0.0	99.6	4	0.0	99.7	0	0.0	99.6
30	29	0.2	99.9	9	0.1	99.8	1	0.2	99.8
31	11	0.1	100.0	9	0.1	99.9	1	0.2	100.0
32	3	0.0	100.0	3	0.0	100.0	0	0.0	100.0
33	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
34	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
35	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
36	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
37	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
38	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
39	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
40	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
41	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
42	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
43	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
44	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
45	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
46	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
47	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
48	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
49	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
50	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
51	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
52	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
53	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
54	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
55	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
56	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
57	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
58	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
59	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
60	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
61	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
62	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
63	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
64	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
65	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
66	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
67	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
68	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
69	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
70	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
71	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
72	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
73	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
74	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
75	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
76	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
77	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
78	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
79	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
80	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
81	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
82	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
83	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
84	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
85	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
86	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
87	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
88	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
89	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
90	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
91	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
92	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
93	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
94	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
95	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
96	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
97	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
98	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
99	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0
100	0	0.0	100.0	0	0.0	100.0	0	0.0	100.0

over

4365 were recorded to be less than 1 sec., 2629 were registered as being 1 sec. in length, 1590 as 2 sec., and so on. Because this table was obtained using the HOURS value (in seconds) taken in each data sample, the first two interval readings are not sufficiently accurate. For greater accuracy at the 0 and 1 sec. intervals, a separate count was kept of the number of searches involved. Of the 6994 values involved in these two intervals (see Table 6-5), 3812 were recorded as being less than 1 search or .1 sec., 369 were less than .2 sec., and so on. Values of 10 searches or more were grouped together as being 1 sec.

Table 6-5 was used to correct the first two intervals of Table 6-4, and the combined result yielded the cumulative distribution over all requests as shown in Table 6-6. Similar distributions were obtained for interactive and heavy users alone. An average think time of 3 sec. was obtained over all requests.

1.1.3 Interrupt Interarrival Time Distribution

The interarrival rate of virtual interrupts within a request was derived from Table 6-7 which classifies the interrupts according to the time interval in which they occurred. These figures include the (12465) interrupts caused by the completion of each request. In

TABLE 6-7 INTERRUPT ARRIVALS (a)

Req. interval (msec)	User no.														Total
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
430	3855	2648	1225	881	1134	584	2055	1195	2437	461	1094	1698	344	14	20041
95	608	771	163	200	119	84	272	57	446	88	491	447	9	3850	
48	82	335	91	121	76	30	32	39	212	60	82	244	1	1453	
24	7	135	77	57	67	27	22	35	139	19	29	221	0	859	
15	80	47	43	25	47	10	27	25	119	19	46	188	0	685	
20	94	480	134	85	98	40	16	43	253	15	47	355	0	1693	
25	20	50	45	41	28	13	7	12	59	14	20	147	0	456	
30	0	103	217	79	66	38	11	31	198	14	31	213	0	1058	
35	0	84	100	33	25	37	1	21	64	7	9	121	0	513	
40	0	54	143	61	67	80	0	31	193	18	31	211	0	943	
45	1	17	94	32	19	31	0	8	93	5	11	86	0	404	
50	3	2	109	97	52	104	0	29	231	66	49	217	0	991	
100	5	45	728	365	98	360	4	192	741	162	398	605	3	3841	
200	5	68	643	424	269	382	0	215	606	338	445	749	3	4245	
300	9	52	270	256	24	291	0	145	609	261	374	155	3	2664	
400	11	57	163	145	24	224	89	113	63	201	247	219	2	1558	
500	5	22	173	176	23	77	0	110	52	243	195	130	1	1207	
600	7	27	108	120	29	56	0	127	52	199	206	156	2	1089	
700	9	25	84	146	13	42	0	167	64	206	206	210	0	1172	
800	3	14	79	52	7	45	0	165	56	228	28	48	0	725	
900	0	27	83	42	7	36	0	79	45	194	39	39	0	552	
1000	0	20	4	46	2	37	0	47	53	194	36	36	0	439	
2000	0	242	0	400	32	195	0	146	427	1898	122	122	0	3462	
3000	0	0	0	317	0	57	0	0	228	630	0	0	0	1232	
4000	0	0	0	397	0	50	0	0	254	596	0	0	0	1297	
5000	0	0	0	243	0	26	0	0	172	285	0	0	0	726	
over 5000	0	0	0	3180	0	26	0	0	272	471	0	0	0	3949	

TABLE 6-8 INTERRUPT ARRIVALS (b)

Req. int'val (msec)	No. requests recorded	No. interrupts minus requests	Per cent.
< 3	5577	20041	14464
3	3598	3850	252
6	1143	1453	310
9	247	859	612
12	116	685	569
15	187	1693	1506
20	67	456	389
25	180	1058	878
30	132	513	381
35	322	943	621
40	31	404	373
45	80	991	911
50	244	3841	3597
100	181	4245	4064
200	59	2664	2605
300	46	1558	1512
400	16	1207	1191
500	14	1089	1075
600	86	1172	1086
700	11	725	714
800	5	552	547
900	3	439	436
1000	71	3462	3391
2000	15	1232	1217
3000	9	1297	1288
4000	9	726	717
5000	16	3949	3933
over 5000			29.7

TABLE 6-9 INTERRUPT ARRIVALS (c)

Req. interval (msec)	NORMAL		INTERACTIVE		HEAVY	
	Per cent.	Cum. dist.	Per cent.	Cum. dist.	Per cent.	Cum. dist.
3	29.7	29.7	40.8	40.8	13.2	13.2
6	0.5	30.3	0.1	40.9	4.4	17.6
9	0.6	30.9	0.2	41.1	0.9	18.5
12	1.3	32.2	1.9	42.9	0.2	18.7
15	1.2	33.3	1.9	44.8	0.6	19.3
20	3.1	36.4	5.1	49.9	0.5	19.8
25	0.8	37.2	1.2	51.0	0.3	20.0
30	1.8	39.0	3.0	54.0	0.4	20.5
35	0.8	39.8	1.2	55.2	0.1	20.6
40	1.3	41.1	1.6	56.8	0.5	21.0
45	0.8	41.8	1.2	58.0	0.1	21.2
50	1.9	43.7	2.5	60.5	1.1	22.2
100	7.4	51.1	9.3	69.8	5.3	27.5
200	8.4	59.5	9.0	78.8	7.2	34.7
300	5.4	64.8	5.5	84.2	6.0	40.7
400	3.1	67.9	2.5	86.7	4.2	44.9
500	2.4	70.4	1.6	88.3	4.1	49.0
600	2.2	72.6	1.5	89.8	3.8	52.9
700	2.2	74.8	1.6	91.4	3.2	56.0
800	1.5	76.3	0.8	92.3	2.4	58.4
900	1.1	77.4	0.7	92.9	2.2	60.6
1000	0.9	78.3	0.3	93.2	2.2	62.8
2000	7.0	85.3	2.8	96.1	18.7	81.5
3000	2.5	87.8	1.0	97.0	5.9	87.4
4000	2.6	90.4	1.1	98.1	5.6	93.0
5000	1.5	91.9	0.7	98.9	2.6	95.6
over 5000	8.1	100.0	1.1	100.0	4.4	100.0

Table 6-8, these completion interrupts were removed to obtain the frequency of virtual interrupts within each interval. For example, of the 20,041 interrupts recorded in the first 3 msec. of all requests, 5577 were attributed to the requests which completed during this time (see Table 6-1). This meant that 14,464 interrupts, or 29.7% of all virtual interrupts, occurred within the first 3 msec. of the requests. Furthermore, 0.5% of all virtual interrupts occurred between 3 and 6 msec. of those requests which were more than 3 msec. in size, and so on.

Notice that the frequencies do not give the distribution for the interarrival time between two interrupts. Rather it gives the time within a request that the next interrupt is likely to occur given the amount of time that request has already executed. In other words, given that a request has already executed 3 msec., there is a 0.5 probability that the next interrupt will occur in the 3 to 6 msec. interval.

The cumulative distribution is given in Table 6-9, along with those calculated separately for interactive and heavy users. From Table 6-9 a distribution of interarrival times between two interrupts was calculated, which was used as input to the simulation model. This was done by generating a number of interrupts, using Table 6-9, at the beginning of each request. These interrupts were

stacked according to their predicted occurrence time. When another interrupt was required, the next interrupt was taken from the stack and the (interarrival) time until its occurrence was calculated.

1.1.4 Interrupt Wait Time Distribution

The sampling technique used to collect data (every .1 sec.) was not adequate to obtain an accurate description of the interrupts and the time necessary to process them. The data obtained showed only that 99% of all interrupts were processed in under 2 searches (.2 msec.).

In lieu of adequate data, an exponential distribution with a mean of 10 msec. was used to describe the interrupt wait time. This distribution was stretched past the 98% point to a maximum wait time of 1 sec. (Thus the mean was slightly over 10 msec.) This was done to allow for the occurrence of a few large I/O interrupts. The maximum wait found in the data was one of 15 searches.

The 10 msec. mean was actually considered to be a high estimate but was used as a worst case figure. However as the number of users increased, it was thought that the 10 msec. figure would become more reasonable since the chances of more conflicts between interrupts would rise resulting in longer waits. Also a greater percentage of interrupts would be due to paging and the average paging operation (from the drum) requires about 12 msec.

1.2 System Description

To model the load caused by the operation of the CP/67 system, it was necessary, and sufficient, to obtain data on the CPTIME used. Three separate time increments made up the total CPTIME (see Chapter IV, Fig. 4.1). The first was the portion of TIMEUSED which was charged to a user for processing each virtual interrupt. For the 61104 interrupts recorded over all samples, a total of 338009 msec. of TIMEUSED was charged. This averages 5.5 msec. per interrupt; between individual samples the average ranged between 3.2 and 8.2 msec. A constant value of 5 msec. was put into the simulation model.

The second value needed was the portion of TIMEUSED which was charged to a user within DISPATCH. A value of .002 msec. was put into the simulation model. This means that, on the average, .002 msec. TIMEUSED was charged to each user each time DISPATCH was executed. This approximate figure was obtained from the data by calculating the total TIMEUSED charged within DISPATCH over all samples and dividing this by both the number of entries into DISPATCH and the average number of users.

The third value needed was the OVERHEAD charged each time DISPATCH was executed. A value of .08 msec. was put into the simulation model to represent the time

taken by DISPATCH to examine one user during it's search for a new user to execute. Again this was an approximate value calculated from the data by dividing the total OVERHEAD used over all samples by both the number of entries into DISPATCH and the average number of users.

1.3 Discussion of Results

An obvious fact which emerges from examination of the measurement results is the dependence of the preciseness of the data upon the measurement technique and the data available within this technique. In particular, this refers to the difficulty of obtaining accurate measurements of small time increments which, in most cases, are the essential quantity being measured. In the four distributions given above, the situation becomes progressively worse until, with the interrupt wait time distribution, the data provided very little information. The .1 sec. sampling interval proved to be very inadequate in this case. Even where the .1 sec. was felt to be sufficient, as in the request size distribution, the preciseness of the data became dependent on the data available (i.e. the fact that requests of less than 3 msec. remained unmeasured even within the system). However since there is always a trade-off between the sampling interval and the amount of data collected, it was felt that the .1 sec. interval

was adequate. In particular, the data collected accomplished its purpose of providing some reasonable and realistic description of the system where no such information, in any detail, was previously available.

2 Simulation Results

2.1 Running Time for Simulation

First concern in the operation of the model was the question of 1) the rate of applying the load to the system during the warmup time, and 2) the running time necessary to collect reliable statistics. Early runs of the model showed that the average interarrival time of requests for service was consistently less than 200 msec. and decreased (to less than 150 msec.) as the number of users increased. Hence users were introduced into the model at a constant rate of one every 200 msec. The object was not to create any unusual loading conditions by introducing the users too rapidly. The warmup time was set at 10 sec. This should allow each user to complete at least one request and to pick the arrival time of his next request. After 10 sec., the requests should be entering the system at a normal (random) rate.

The large amount of computer time required to run the model prevented simulation of large time intervals. The aim, then, was to simulate only long enough so that

early extreme fluctuations from the random number generator would stabilize. During a 2 min. run, sample statistics on the system's operation (particularly the average request size) were taken every 5 sec. On the basis of these samples, it was decided that 30 sec. of running time was sufficient to achieve this objective. The final results of the simulation model, then, are based on a simulated run of 30 sec. To simulate 23 normal users for this 30 sec. required between 9 and 10 min. of computer time. Fewer users required less computer time. For example, 12 normal users only required 2.5 min. of computer time.

2.2 Validation of the Simulation Model

As in any simulation project, there remains the final step of model validation before the results from the simulation model can be considered reliable and useful. Of all the problems associated with computer simulation, that of verifying the simulation model is perhaps most elusive [24]. Some of the problems encountered in this project are discussed in Sec. 2.4.

In addition to the CPU utilization figures (WAITTIME, CPTIME, and PROBTIME), the two ratios of CPTIME to PROBTIME (CP/PROB) and TIMEUSED to PROBTIME (TU/PROB) were used in comparing the operation of the DISPATCH model with that of the real system. These last two values were used since they reflect that part of the total system operation for which DISPATCH is responsible. The CP/PROB ratio is also a measure of the CPU utilization in relation to the amount of processing requested.

2.2.1 Data Used for Validation

Table 6-10 gives some of the overall results observed in the measurement data that were used for validation of the model. The table summarizes the number of users, WAITTIME, CPTIME, PROBTIME, CP/PROB ratio, and the TU/PROB ratio for each of the 14 measurement samples as well as the overall averages. The table shows a lot of variation with the number of users varying from 5 to 15, the WAITTIME from 0.8% to 91.8%, CPTIME from 5.8% to 71.9%, PROBTIME from 2.4% to 55.8%, CP/PROB ratio from .4 to 3.9 and the TU/PROB ratio from 1.2 to 5.4. A CP/PROB ratio of 3.9 and a TU/PROB ratio of 5.4 indicates that for each second spent in problem mode, the system uses 3.9 seconds for overhead (CPTIME), and the user is charged for 5.4 seconds (TIMEUSED). In the overall average the total time was divided approximately equally between the three times (WAIT, CP, PROBTIME). The overall CP/PROB and TU/PROB ratios were 1.1 and 1.8, respectively.

Of particular interest is the seven samples with high CPU utilization (i.e. WAITTIME less than 16%), since these samples (no. 1, 3, 5, 8, 10, 11, 12) represent the system under loaded conditions. Although all these runs have at least 11 users, their average is only 12.1 as compared to the overall average of 10.7. Thus the number of users is not a good indication of the load on the system, unless there is some knowledge of the nature of the input these users are representing.

TABLE 6-10 INITIAL MEASUREMENT RESULTS

	Sample no.														Overall average	Ave. over 7 loaded samples
	1	2	3	4	5	6	7	8	9	10	11	12	13	14		
No. Users	13	9-11	12-13	9-12	10-12	5-11	7-11	10-15	5-7	9-12	10-13	14-15	10	11	10.7	12.1
% WAIT	9.6	57.9	9.7	72.6	15.6	48.4	39.8	2.7	91.8	9.6	.8	14.0	37.2	46.8	32.1	8.9
% CP	71.9	19.3	46.9	15.5	30.6	30.2	31.9	58.2	5.8	65.8	49.5	30.1	24.7	16.4	35.9	50.4
% PROB	18.5	22.8	43.4	11.9	53.7	21.4	28.3	39.1	2.4	24.6	49.7	55.8	38.0	36.8	32.1	40.7
CP/PROB	3.9	0.8	1.1	1.3	0.6	1.4	1.1	1.5	2.4	2.7	1.0	0.5	0.7	0.4	1.1	1.2
TU/PROB	3.2	2.7	4.6	2.0	2.6	1.5	5.2	5.4	2.5	1.5	1.2	1.2	4.6	5.0	1.8	1.5

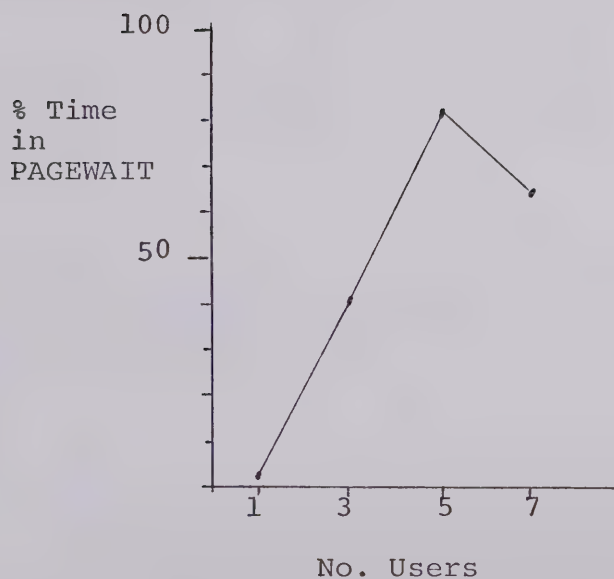
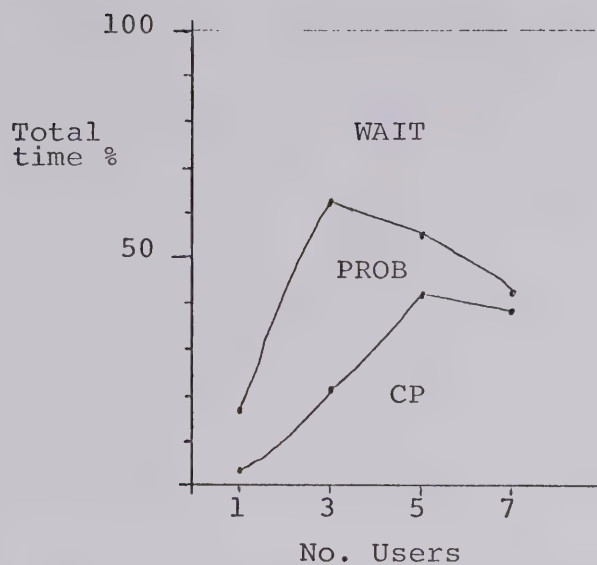
This is also illustrated by examining the wide variation of the CP/PROB ratio within these seven samples. The CP/PROB ratio ranged from .6 to 3.9, almost the full range found over all samples. This indicates that the input which was represented by the users in these samples differed significantly although they all produced a loaded condition in the system. The high CP/PROB ratio probably resulted from a large number of interactive users, or those requiring large amounts of paging, whereas more compute-bound users would result in the lower CP/PROB ratios.

The TU/PROB ratio for these samples varies over the entire range with an average of 1.5 compared to the overall average of 1.8. The wide variation in TU/PROB is probably caused by large variations in the amount of I/O the users are doing - high ratios from heavy I/O.

Since knowledge of the user input is such a vital consideration, another method of validating the simulation model is to set up a controlled load situation in which one knows exactly what each user is doing. Such an experiment was developed by G. Neufeld [25] for CP/67. In this experiment each test user was executing an identical program involving continuous matrix multiplications with a controlled amount of paging. System load was controlled

by increasing the number of test users. Using this experiment, additional test measurements were made on the CP/67 system in an attempt to obtain figures with which to validate performance of the model and, at the same time, examine performance of the system under a controlled load situation. The results obtained are illustrated in Fig. 6.1. (It should be stressed that these users are each equivalent to approximately 5 normal users.)

Best performance is, of course, indicated with one user. At 3 users, the observed response dropped noticeably although performance was still acceptable. The CP/PROB ratio was .57. However the user followed (which was typical of all other users) was found to be in PAGEWAIT 40% of the time. At 5 users the performance of the system was extremely poor; the CP/PROB ratio jumped to 3.5 and the user followed was in PAGEWAIT 83% of the time. At 7 users the system was barely responding. Only 4% of the time was spent in problem state. The CP/PROB ratio leaped to 13.5. WAITTIME increased while PAGEWAIT time decreased indicating such severe congestion that the user was not being served enough to even demand pages. (The system is thrashing, a condition discussed in Chapter II, Sec. 2.2.2.2.)



No.Users	% WAIT	% CP	% PROB	CP/PROB	TU/PROB	% PAGEWAIT
1	83	3	14	.20	1.0	1.3
3	39	22	39	.57	1.3	40
5	46	42	12	3.5	3.0	83
7	58	38	4	13.5	5.7	65

Fig. 6.1 Controlled Test Load Results

2.2.2 Evolution of the Simulation Model

Although problems were found in comparing load conditions (and hence performance) between the model and the real system, the model showed it's flexibility in it's ability to be adapted to give the desired performance for a given load. Thus, once agreement on a load definition is reached, the model can be adjusted such that it's performance will match that of the real system under the same load.

For example, in the process of validating the model, many tests and alterations were made. Initial runs of the model were able to handle up to 31 users with no sign of overloading, as the following performance figures illustrate.

% WAIT	% CP	% PROB	CP/PROB	TU/PROB	Ave.Reg. Size (msec)	Response Request
14.2	40.7	45.0	.90	1.6	39.3	8.0

The model was then changed by making the number of interrupts generated a function of the number of users on the system. A multiplication factor of $(\frac{N}{15})^2$ was added to the number of interrupts assigned a request, where N is the number of users. The number 15 was arbitrarily chosen as the "loading point" since it appeared to be the point at which serious conjection began to occur in the paging

activity of the system. (A lower figure could be used to indicate larger program sizes.) The results of this change were as follows.

With 23 users:

% WAIT	% CP	% PROB	CP/PROB	TU/PROB	Ave.Reg. Size (msec)	Response Request
31.8	41.4	26.7	1.55	2.1	29.1	9.7

With 26 users:

% WAIT	% CP	% PROB	CP/PROB	TU/PROB	Ave.Reg. Size (msec)	Response Request
20.3	49.0	30.6	1.59	2.2	44.3	7.3

Although the increased number of interrupts brought the CP/PROB and TU/PROB ratios up to a realistic level, the response time indicated that the system was still under-loaded with 26 users.

The model was again altered to allow more than one interrupt at the same time. Previously, duplicate interrupts (generated to occur at the same time within a request) were discarded. It was felt that this change significantly improved the model which now gave the following results.

With 23 users:

% WAIT	% CP	% PROB	CP/PROB	TU/PROB	Ave.Reg. Size (msec)	Ave.resp. Ave.req.
3.5	67.6	28.7	2.35	2.8	48	14.9

Reaction of the model under load was now reflected in the response time. A final change was made in an attempt to increase the WAITTIME since it was now considered too high as compared with the results of the test load measurements (Fig. 6.1).

To increase the WAITTIME, the interrupt wait time was increased. The same $(\frac{N}{15})^2$ multiplication factor was used in increasing the time required to process each interrupt. The 15 user "loading" point was used again for the reason given before and also since it is at this point that CP/67 would normally start using the disk for paging, thus resulting in longer page transfer times. (Again this multiplication factor is arbitrary and could easily be altered to reflect either more or less congestion as a function of the number of users.) The results of this change are shown below.

With 23 users:

% WAIT	% CP	% PROB	CP/PROB	TU/PROB	Ave.Reg. Size (msec)	Response Request
14.3	54.8	30.7	1.78	2.4	21.8	39.8

At this point, the performance of the model was considered to be a good approximation of the performance of the real system as was measured in the data. (Comparison of the model results with the measurement results is contained in the following section.)

The flexibility of the model was thus illustrated. The model can be adapted to yield the desired performance given a defined load. The model was equally as flexible in its ability to define different input conditions to obtain the desired load. By defining users as normal, interactive, and heavy, the model was able to alter the request size distribution and the arrival rate of requests. Interactive users have smaller requests with larger think times (see Tables 6-3 and 6-6), while heavy users have larger requests with smaller think times. The number of users in each of the three classes can be varied to fit the desired load.

2.2.3 Results from the Model and Validation

Table 6-11 gives the results of some runs of the model in which the number and class of users were varied to obtain different load conditions. Table 6-12 summarizes the results of all the measurement data obtained and can be used for comparison with Table 6-11. For the reasons given in Sec. 2.4, response time measured in the data was not used for comparison purposes.

The first four rows of Table 6-11 and Fig. 6.2 show the performance of the model as the load (no. of users) increases. Model performance at 12 and 15 users is very close to that given by the average obtained over the 7 light load samples of the initial measurement data (see Fig. 6.3). The 12 user level was used as a comparison point since it was considered to be the approximate

TABLE 6-11 MODEL RESULTS

No. Users	% WAIT	% CP	% PROB	CP/PROB	TU/PROB	Ave. Req. Size (msec)	Response Request
12 normal	63.7	16.3	19.9	.81	1.6	38.0	3.1
15 "	52.8	28.4	18.7	1.51	2.2	29.8	7.4
20 "	28.6	43.6	27.2	1.57	2.2	25.4	22.7
23 "	14.3	54.8	30.7	1.78	2.4	21.8	39.8
15 inter.	58.5	25.7	15.7	1.63	2.2	24.3	6.5
15 heavy	.3	30.0	69.5	.43	1.3	595.2	5.5
15 normal, 5 heavy	.3	48.8	50.7	.96	1.8	116.5	8.5
10 normal, 5 heavy	14.4	33.6	51.8	.64	1.5	95.5	5.5
15 normal, 2 heavy	27.8	29.8	42.3	.7	1.5	71.2	5.7
20 normal*	46.9	40.7	12.2	3.31	3.4	17.6	15.6
20 inter.*	47.4	45.8	6.7	6.8	6.0	7.7	32.7

TABLE 6-12 MEASUREMENT RESULTS

		% WAIT	% CP	% PROB	CP/PROB	TU/PROB
Table 6-10 Overall ave.		32.1	35.9	32.1	1.1	1.8
Ave. over 7 loaded samples		8.9	50.4	40.7	1.2	1.5
Ave. over 7 light samples		56.3	20.5	23.1	1.0	2.2
Controlled test load (Fig. 6.1)	1 user	83	3	14	.2	1.0
	3 users	39	22	39	.57	1.3
	5 users	46	42	12	3.5	3.0

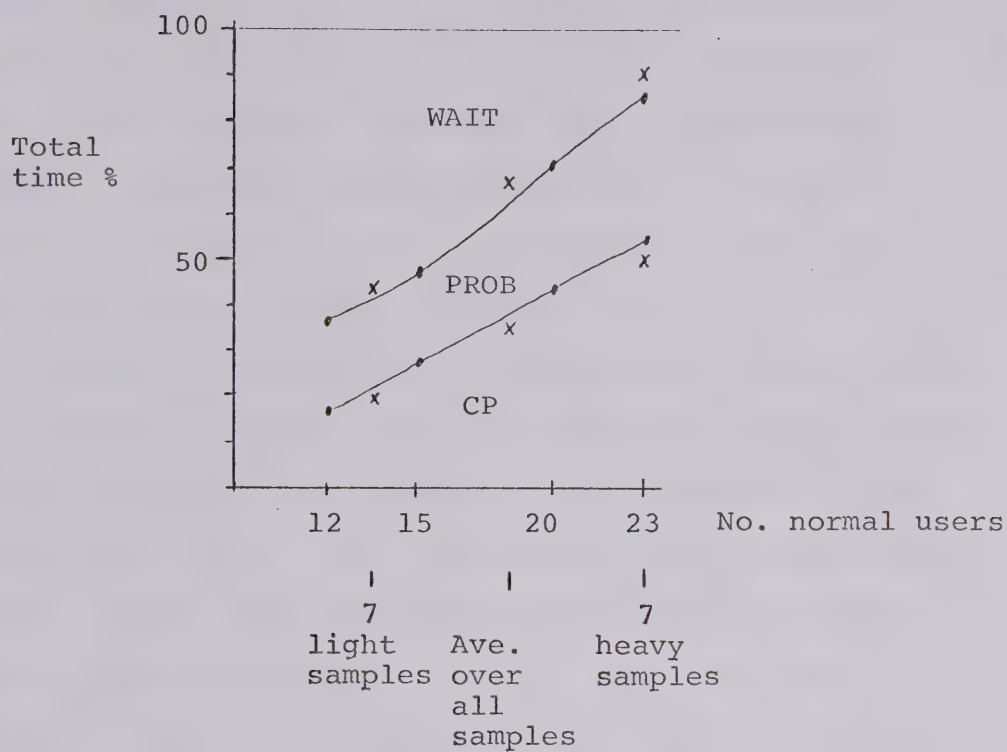


Fig. 6.2 Validation Comparisons (a)

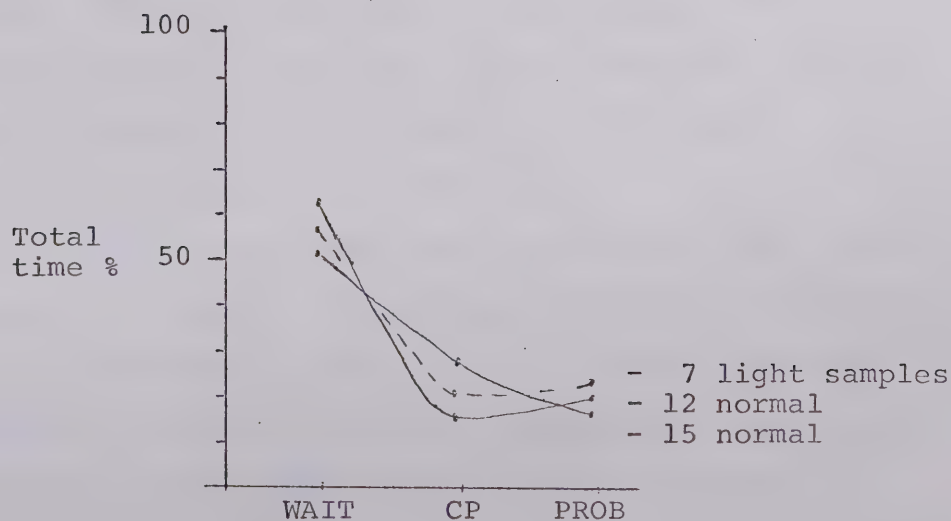


Fig. 6.3 Validation Comparisons (b)

average number of users on the system when the measurement figures were obtained. The response time at 12 and 15 users indicates that the model is not heavily loaded. Furthermore, model performance at 20 and 23 users is very close to that given by the average obtained over the 7 loaded samples (see Fig. 6.4).

Figures on the model's performance at heavier loads also rose in agreement with the measurement figures found in the controlled test load situation, although without the extreme values. The model at 20 and 23 users appears to fit in the 3-5 user range of the test load users. The two model runs marked with an * were run using a different random number generator with which to select the request sizes. The result was a much lower average request size which put a highly interactive load on the system. The corresponding poor performance figures show how the extreme values seen in the measurement data can also be reached with the model. The 20 normal * users correspond to the 5 test load users.

Further sample runs are included in Table 6-11 for comparison purposes. For example, 15 normal and 5 heavy users correspond to sample no. 11 in Table 6-11 and illustrate how the load can be adjusted to obtain the extremely low WAITTIME figures seen in the data.

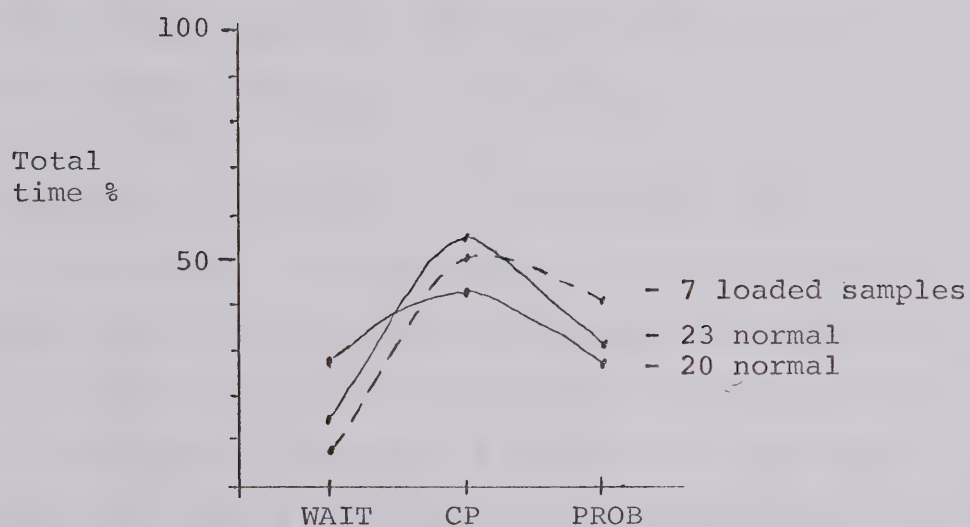


Fig. 6.4 Validation Comparisons (c)

The random number generator is still making comparison difficult. From observing the results and from experimenting with the model, it can be seen that both the CP/PROB and TU/PROB ratios are functions of the request size (i.e. functions of the ratio of interactive to computing requests). However the author is quite confident that if the average measured request size of 46 msec were simulated, the CP/PROB and TU/PROB would be in very close agreement with the overall averaged measured results (see Fig. 6.5 and 6.6).

Examination of Table 6-11 also reveals some interesting conclusions. In particular, it would appear that although a time-sharing system is designed for interactive use, it is desirable to have some heavy user background load. For example, comparing 15 interactive users with 10 normal and 5 heavy users, it can be seen that the 10 and 5 load achieves both better utilization of the processor time and also a better response time.

The model was thus considered to be a good representation of the real system. The model was adjusted such that a load of 20-23 normal users represented a relatively heavy load as compared with the test load measurements. This load was then used in further testing of ways to improve system performance under heavy loads.

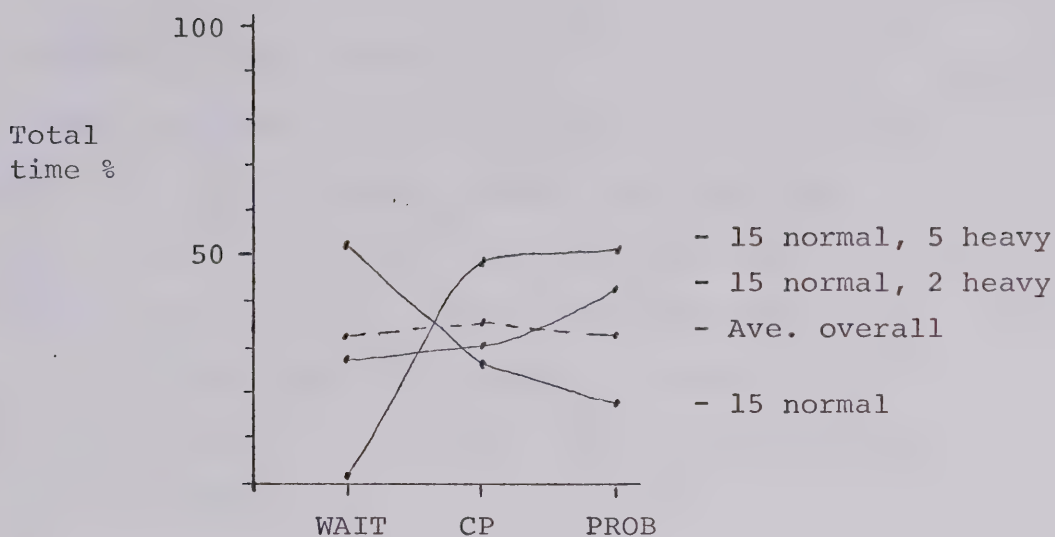


Fig. 6.5 Validation Comparisons (d)

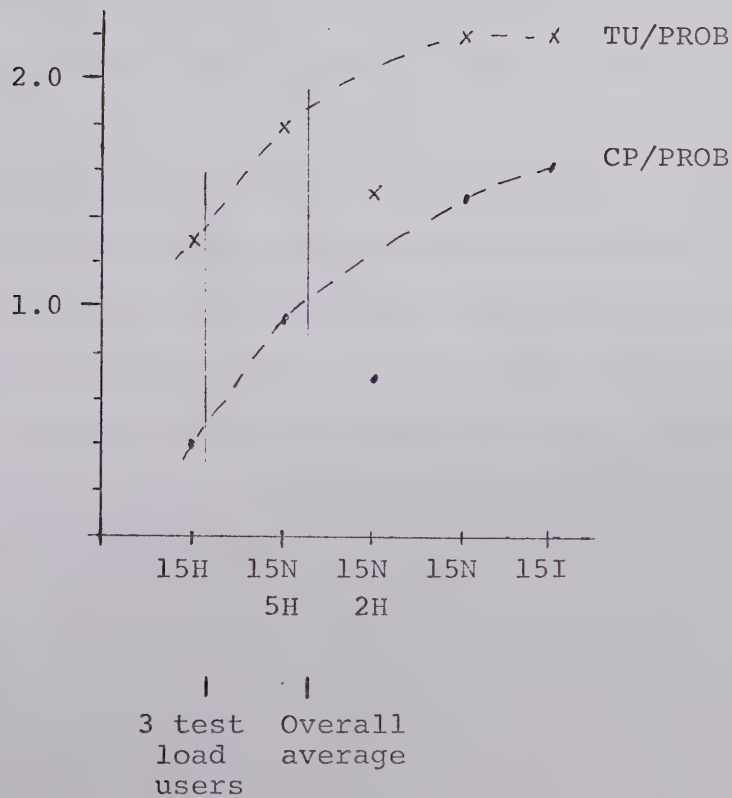


Fig. 6.6 Validation Comparisons (e)

2.3 Results of Model Testing

Once satisfied that the model was in fact a good representation of the system, several test runs were executed in an effort to find ways in which system performance might be improved through alterations in the job scheduler. The first parameter tested was the time-slice. The following results were obtained using a load of 23 normal users.

Time-slice	% WAIT	% CP	% PROB	CP/PROB	TU/PROB	Ave.Reg. Size (msec)	Response Request
12 msec.	28.7	58.3	12.8	4.53	4.3	16.3	27.0
50 msec.	28.9	58.1	12.8	4.51	4.3	16.2	26.3
500 msec.	28.8	58.2	12.8	4.52	4.3	16.2	26.8

The figures clearly showed that the changes in the time-slice had no effect on the performance of the system.

The average request size of 16 msec. would indicate that the load represented in these runs was highly interactive. Additional tests were made under extremely high compute-bound conditions. The following results were obtained with a load of 15 heavy users.

Time-slice	% WAIT	% CP	% PROB	CP/PROB	TU/PROB	Ave.Reg. Size (msec)	Response Request
12 msec.	0	23.1	66.8	.49	1.3	355.7	13.0
50 msec.	0	29.8	70.2	.42	1.3	360.1	11.6
500 msec.	0	29.3	70.5	.41	1.3	360.4	10.2

Under these load conditions, best performance is observed with a 500 msec. time-slice. For large requests, the 500 msec. time-slice creates less interference than the smaller time-slices, thus better response. The 12 msec. figures show some difference in performance, although the 50 msec. figures show little significant difference in performance with the 500 msec. figures. Thus, although a time-slice of 500 msec. might be recommended under heavy compute-bound conditions, the value of 50 msec. appears to be satisfactory.

Additional measurements were made on a more normal load situation. The following results were obtained with 15 normal and 5 heavy users.

Time-slice	% WAIT	% CP	% PROB	CP/PROB	TU/PROB	Ave.Reg. Size (msec)	Response Request
12 msec.	2.4	53.4	44.1	1.2	1.9	103.2	6.6
50 msec.	21.4	48.8	29.7	1.63	2.2	61.3	7.9
500 msec.	15.7	50.5	33.6	1.49	2.1	59.7	8.5

Best response is seen with a time-slice of 12 msec. .

(The higher request size should not be a factor in this as demonstrated in the immediately preceding figures.)

There is little significant difference between performance with the 50 msec. time-slice and the 500 msec. time-slice. The smaller time-slice appears to improve the response of smaller requests when they are in competition with large requests. Thus overall performance is improved.

From these three test runs, it would be concluded that:

- a) the time-slice has no effect on highly interactive loads.
- b) a large time-slice is recommended for heavy compute-bound loads.
- c) small time-slice will improve performance under some conditions where small requests are in competition with large requests.

In general, however, it would be concluded that the time-slice has little effect on performance and that the 50 msec. time-slice which presently exists is quite adequate.

Further tests were carried out by varying the maximum number of users allowed in queue. The normal values used in the operation of CP/67 were a maximum of

9 users in Q_1 and 6 users in Q_2 . The results of these tests are given below; a load of 23 normal users was again used.

Max. Q size Q_1-Q_2	% WAIT	% CP	% PROB	CP/PROB	TU/PROB	Ave. Req. Size (msec)	Response Request
9 - 6	14.3	54.8	30.7	1.78	2.4	21.8	39.8
6 - 6	18.9	61.8	19.1	3.22	3.4	28.6	23.5
6 - 3	12.0	64.4	23.4	2.75	3.0	27.4	29.1

Results showed a definite improvement in the response time of the system at the 6-6 level compared with the 9-6 level. However there was a corresponding drop in the efficiency of the CPU utilization (i.e. the CP/PROB and TU/PROB ratios jumped). This apparent contradiction - better response with less time in problem mode - is probably a result of short requests being serviced more rapidly while the relatively few longer requests wait. Tests in the 6-3 level showed a reversal of the above trend with some improvement in CPU utilization and an increase in response time. It appears that the queue size parameters are an excellent means of reaching almost any desired trade-off between response time (service to the user) and CPU utilization.

Since response time must be considered the primary performance measure, the results would indicate that a lower maximum number of users allowed in Q_1 would improve the performance of the system.

The usefulness of the simulation model has thus been illustrated. The model can be used as a valuable tool to aid in making constructive recommendations for improving the operation of a system.

2.4 Discussion

Although validation of the model was considered accomplished through the good comparisons obtained between the performance of the model and that of the real system, the project pointed out the problems inherent in the validation of a simulation model, particularly that of a time-sharing computer system. These problems exist in 1) deriving a basis of comparison (i.e. defining an input load) and 2) obtaining comparative measurement data from the system.

The main difficulty in defining the load on a time-sharing system is derived from the difficulty of characterising user requests (or user input). Each user is working on a completely separate problem from all other users. Thus the load he represents to the system, in terms of request sizes and rate of arrival,

is different from all others. Also each request is different in terms of the number of pages required, the number and size of I/O operations, etc. Thus, in a normal situation, the input load to a time-sharing system is made up of a vast conglomeration of different user requests. To arrive at a definition of an "average" or "heavy" load is obviously very difficult.

In this project, load was defined by measuring the load represented by 14 common users of CP/67 and averaging their combined data to obtain one "normal" user. Because of the large volume of data that was necessary to get the desired measurement accuracy, only one user could be monitored at a time. Hence it was not possible to relate accurately the performance of the system to the total user input. Furthermore the grouping of the 14 users as normal, interactive, and heavy, and averaging over each group had the effect of smoothing any extreme loading conditions. Since it has been found (by applying a known load to the real system) that CP/67 is very intolerable of overloads, the effects of smoothing the input to the model probably increased its performance significantly.

The problems encountered in obtaining precise measurement data have already been introduced in Sec.1.1.3. The problem of too slow a sampling rate can be appreciated

from observing the large percentage of items that appear in the smallest time interval of Tables 6-3, 6-6, and 6-9. The distribution over this initial interval is highly significant, yet is unknown. For example, 70% of all responses were recorded as less than one search. Thus all we know is that these requests received a response in less than 100 msec. It was for these reasons that the measured response time was not used for comparison with the model.

Although more detailed and accurate data would have been preferred, it was possible to make good comparisons between the model and the real system with the data available. This could be done even when some parts of the system (e.g. interrupt wait time, paging), which were not measurable at all, had to be allowed for in a somewhat arbitrary fashion. The model was able to achieve its objective of testing the influence of the job scheduler on the performance of the system.

Figures from test runs showed interesting results. It was concluded that the time-slice of 50 msec., which currently exists in CP/67, was a good value to use in the attempt to balance the service given to interactive and heavy users. It was found that by lowering the

maximum number of users allowed in Q_1 , a better response time could be achieved. This is perhaps one change which could be recommended on the real system. The results also showed that some heavy background users are desirable to allow more efficient utilization of the CPU as compared with utilization under a load of highly interactive users.

CHAPTER VII

CONCLUSIONS

In general, simulation has two main purposes. The first, which has been emphasized in this thesis, is to provide a basis for predicting how the simulated system will perform under varied conditions and for determining which conditions are necessary for optimum performance.

In this respect, the project has yielded several interesting results concerning the operation of CP/67. Although CP/67 may be very useful to a relatively few programmers who each require a virtual machine (e.g. to develop new operating systems), it appears from the results obtained that major improvements are necessary to improve the capacity of CP/67 before it could handle the computing requirements at U of A. For instance, the measurement data showed that 12 to 15 users sometimes overloaded the system although this is a relatively light load. The normal capacity is believed to be about 20 users and the system would probably be so overloaded with 25 users that it's response would be unacceptable. Furthermore, the trials with test load users showed that 5 special users (which are equivalent to about 20 normal users) overloaded the CP/67 system and 7 users put it into a state from which it cannot recover.

The testing which was carried out with the model has shown the value of simulation in analysing time-sharing computer systems. Results of the model showed that the time-slice has little effect on the performance of the system. Only under extreme conditions were differences in system performance measured as a function of the time-slice. It was concluded that the 50 msec. value which presently exists in the CP/67 system should remain, since it seemed to represent a reasonable mid-point in the trade-off which must be considered between servicing interactive and heavy requests. As a result of the tests, it was recommended that a smaller limit should be set on the maximum number of users allowed in Q_1 . A smaller limit seemed to achieve better response times. The queue sizes were also seen to be a good means of reaching any desired trade-off between good response time and good CPU utilization. Finally, the results showed that it is desirable to maintain a background load of some heavy users to improve the efficiency of the system over a straight interactive load.

Among the overall achievements attained in the project was the development of a sophisticated data gathering package with which to measure the CP/67 system. The technique yields much more detailed information on the operation and performance of the system than was

previously available. Along with this, an extensive data analysis program was developed to interpret the data produced and to organize the results in a format suitable for input to the model.

The development of the model is, of course, another significant achievement. The model can be used as the basis of further research into the operation of CP/67.

A second, more basic, purpose of simulation is its use as a learning tool. To explain: in Chapter V (Sec. 1) we described a simulation model as being composed of two parts; a model of the input environment and a model of the system operation. Because a first-hand knowledge of the operation of a system is usually a prerequisite to simulating that system, it is the input description which becomes the focal point in the work involved in building a valid model. It often occurs, at some point in the project, that a greater knowledge of some aspect of the system is necessary or is uncovered. As such, simulation becomes a valuable learning tool, forcing the analyst into a greater appreciation and understanding of the system under study.

Realizing this aspect of simulation, one of the initial reasons for undertaking a project of this nature was a desire to gain some insight into the operation of

modern computer software systems. In this respect, the project was an unquestionable success. As Naylor et al. [24] states; "The experience of designing a computer simulation model may be more valuable than the actual simulation itself".

Many lessons were learned about the procedures involved in developing a simulation model. Simulation involves many steps: collection and analysis of data, development of the model, programming the model, validating the model and analysing the simulation data to mention the basic ones. The greatest overall lesson learned was that no one step is separate from the rest, and that consideration of all steps must be kept in mind at all times. For example, formulation of the model must be made on the basis of the data available which in turn is dependent on the method used to obtain data. Validation of the model must be considered during collection of the data since validation cannot be achieved without adequate measurement figures.

Also the project pointed out the lack of a scientific approach in the "art" of evaluating computer systems. However the author feels that this is inherent in the very nature of the field, and is not likely to change significantly in the future. Each computer system is a complex "individual", and the evaluation of each must be approached individually, usually with a good deal

of trial-and-error methods involved. The most valuable prerequisite for undertaking any systems evaluation project is some previous experience in this area, not to mention some practical experience with computer systems. Again the author would like to list the experience gained in this area as one of the successes of this project.

One area in which room for possible future improvements can be seen is in the methods of obtaining measurements from computer systems such as CP/67. Hopefully, as the need for methods of measuring and evaluating the performance of the modern complex computer systems has been shown, system designers will expend more effort into developing the means of monitoring and measuring these systems. At present, the system analyst is required to expend a great portion of his time in obtaining measurement data; the lack of adequate tools for measuring the system forces him to develop his own.

The sampling technique used, although adequate for our purposes, would not be adequate for a detailed analysis of a time-sharing system. Smaller sampling intervals may be possible but would only yield an unwieldy volume of data.

There is, of course, much room for further use of the model. Although some testing was done with the model, the project has more or less only pointed out how the model can be used. Further tests could have been done if more time and more computer time were available. The model developed however has been shown to be a good representation of the real system and is ready for further use.

For future work, the logic of the simulation model could be expanded to better represent the role of the operating system in the management of the computer's space resources. In particular, the role of paging in the performance of the system should be investigated. This, however, would be a very involved undertaking, more than equivalent to this research. One of the prerequisites for such a study would be a much better method of measuring the operation of the system.

Although it is very difficult to define the load on a time-sharing system, the controlled test load experiment is considered as the best method available. Further use of the model might involve designing such an experiment specifically to use in comparison with the model.

In conclusion, the success of the project has been two-fold. A simulation model has been developed which can be used as a valuable tool in evaluation of the CP/67 time-sharing system. Also the project has been a valuable learning experience, both in understanding the complexities of modern computer operating systems, and in learning the procedures, and difficulties, involved in simulation.

REFERENCES

- 1 Bishop, O.S., A Queueing Analysis of the CP/67 Time-Sharing System, Master's Thesis, University of Alberta (1969).
- 2 Calingaert, P., "System Performance Evaluation: Survey and Appraisal", Comm. ACM, Vol. 10, No. 1 (Jan. 1967).
- 3 Cantrell, H.N., and Ellison, A.L., "Multiprogramming Performance Measurement and Analysis", Proc. SJCC, Vol. 32 (1968).
- 4 Coffman, E.G., "Analysis of a Drum I/O Queue Under Scheduled Operation in a Paged Computer System", J. ACM, Vol. 16, No. 1 (Jan. 1969).
- 5 Coffman, E.G., and Kleinrock, L., "Feedback Queueing Models for Time-Shared Systems", J. ACM, Vol. 15, No. 4 (Oct. 1968).
- 6 Control Program-67/Cambridge Monitor System Manual, IBM Cambridge Scientific Center, Mass. (1968).
- 7 Corbato, F.J., Merwin-Daggett, M., and Daley, R.C., "An Experimental Time-Sharing System", Proc. SJCC, Vol. 21 (1962).
- 8 Deniston, W.R., "SIPE: A TSS/360 Software Measurement Technique", Proc. ACM 24th Nat'l Conf. (1969).
- 9 Denning, P.J., "Effects of Scheduling on File Memory Operations", Proc. SJCC, Vol. 30 (1967).
- 10 Denning, P.J., "The Working Set Model for Program Behavior", Comm. ACM, Vol. 11, No. 5 (May 1968).
- 11 Denning, P.J., "Thrashing: Its Causes and Prevention", Proc. FJCC, Vol. 33, Part 1 (1968).
- 12 Everett, R.R., Zraket, C.A., and Benington, H.D., "SAGE - A Data-Processing System for Air Defence", Proc. EJCC (1957).
- 13 Fine, G.H., and McIsaac, P.V., "Simulation of a Time-Sharing System", Management Science, Vol. 12, No. 6 (Feb. 1966).

- 14 Fine, G.H., Jackson, C.W., and McIsaac, P.V., "Dynamic Program Behavior Under Paging", Proc. ACM 21st Nat'l. Conf. (1966).
- 15 Gatha, A.K., Statistical Evaluation of CP/67, A Time-Sharing System, Master's Thesis, University of Alberta (1970).
- 16 Hellerman, H., "Complementary Replacement - A Meta Scheduling Principle", Proc. ACM 2nd Symposium on O.S. Principles (1969).
- 17 IBM Form H20-0304-3, "General Purpose Simulation System/360 Introductory User's Manual".
- 18 IBM Form H20-0326-2, "General Purpose Simulation System/360 User's Manual".
- 19 Joseph, M., "An Analysis of Paging and Program Behavior", Computer J., Vol. 13, No. 1 (Feb. 1970).
- 20 Karush, A.D., "Two Approaches for Measuring the Performance of Time-Sharing Systems", Proc. ACM 2nd Symposium on O.S. Principles (1969).
- 21 Kuehner, C.J., and Randell, B., "Dynamic Storage Allocation Systems", Comm. ACM, Vol. 11, No. 5 (May 1968).
- 22 Martin, F.F., Computer Modeling and Simulation, John Wiley and Sons, Inc. (1968).
- 23 McCredie, J.W., "Measurement Criteria for Virtual Memory Paging Rules", Proc. ACM 24th Nat'l. Conf. (1969).
- 24 Naylor, T.H., Balintfy, J.L., Burdick, D.S., and Chu, K., Computer Simulation Techniques, John Wiley and Sons, Inc. (1966).
- 25 Neufeld, G., Unpublished doctoral research, University of Alberta (1970).
- 26 Nielsen, N.R., "The Simulation of Time-Sharing Systems", Comm. ACM, Vol. 10, No. 7 (July 1967).
- 27 Nielsen, N.R., "An Approach to the Simulation of a Time-Sharing System", Proc. FJCC, Vol. 31 (1967).
- 28 Oppenheimer, G., and Weizer, N., "Resource Management for a Medium Scale Time-Sharing Operating System", Comm. ACM, Vol. 11, No. 5 (May 1968).

- 29 Pinkerton, T.B., "Performance Monitoring in a Time-Sharing System", Comm. ACM, Vol. 12, No. 11 (1969).
- 30 Roek, D.J., and Emerson, W.C., "A Hardware Instrumentation Approach to Evaluation of a Large Scale System", Proc. ACM 24th Nat'l. Conf. (1969).
- 31 Rosin, R.F., "Supervisory and Monitor Systems", ACM Computing Surveys, Vol. 1, No. 1 (Mar. 1969).
- 32 Sackman, H., "Time-Sharing versus Batch Processing; the Experimental Evidence", Proc. SJCC, Vol. 32 (1968).
- 33 Scherr, A.L., An Analysis of Time-Shared Computer Systems, MIT Press, Cambridge (1967).
- 34 Schwartz, J.I., Coffman, E.G., and Weissman, C., "A General Purpose Time-Sharing System", Proc. SJCC, Vol. 25 (1964).
- 35 Schrage, L.E., and Miller, L.W., "The Queue M/G/1 with the Shortest Remaining Processing Time Discipline", Op. Res., Vol. 14 (1966).
- 36 Shulman, F.D., "Hardware Measurement Device for IBM System/360 Time-Sharing Evaluation", Proc. ACM 22nd Nat'l. Conf. (1967).
- 37 Teichroew, D., and Lubin, J.F., "Computer Simulation - Discussion of the Technique and Comparison of the Languages", Comm. ACM, Vol. 9, No. 10 (Oct. 1966).
- 38 Walter, C.J., Bohl, M.J., and Walter, A.B., "Fourth Generation Computer Systems", Proc. SJCC, Vol. 32 (1968).

APPENDIX A

CP/67 VIRTUAL ENVIRONMENT

1 Addressing and Paging

There are four basic tables used by CP/67 for addressing information.

- a) SEGTABLE: Each user has one segment table. Each entry in the segment table points to a corresponding page table. The address of SEGTABLE is kept in the UTABLE of each user.
- b) PAGETABLE: For each virtual page belonging to a user, there exists an entry in his PAGETABLE. Each entry contains the real address of that page if it is in core, and the in-core indicator for that page.
- c) SWPTABLE: Similarly for each virtual page belonging to a user, there exists an entry in his swap table. Each entry contains the direct access storage device (DASD) address of the corresponding page when it is not in core. It also contains an in-transit indicator for that page.
- d) CORETABLE: The core table consists of an entry for each page of real core. The physical location of a page in core is

determined by the relative position of its corresponding entry in the core table.

Each entry contains a pointer to the SWPTABLE entry corresponding to the virtual page currently occupying the real page.

As well it contains an in-transit indicator for that page, a LOCK indicator and the UTABLE address of the page's user.

Fig. A.1 illustrates the mappings used between these tables during addressing and paging. The four high-order bits of an address provide a pointer to the appropriate page table by providing a displacement from the beginning of the segment table. The next eight bits of the address provide a displacement from the beginning of this page table, thus pointing to the appropriate page. The twelve low-order bits provide a displacement from the beginning of the page, thus completing the translation of virtual address to real address.

If the page is found not to be in core, as indicated in its PAGETABLE entry, the entry for that page is found in the SWPTABLE. If the page is not already in transit, CP/67 proceeds to bring it into core. First the CORETABLE is examined to select a page to be removed. Simply stated, CP/67's page replacement algorithm will

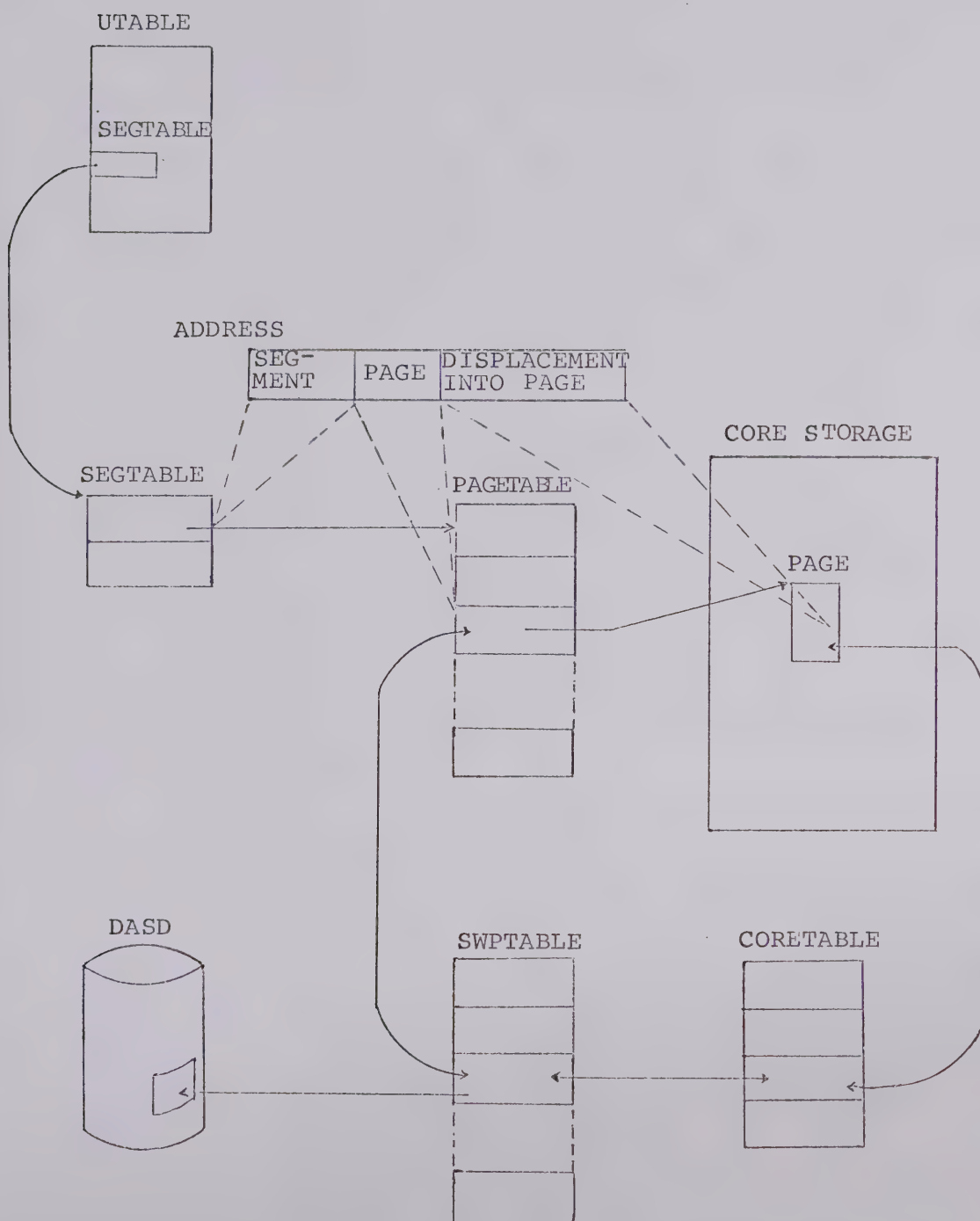


Fig. A.1 Addressing and Paging Tables

select a page for removal if it has not been used or if the user associated with that page is not in Q_1 or Q_2 of the scheduler. LOCKed or in-transit pages are exempt.

When a page has been selected to be removed, its PAGETABLE entry is marked not-in-core and, if necessary, it is swapped back into its DASD storage location. The PAGETABLE entry of the missing page is then given this real address and marked not-in-core. The corresponding CORETABLE and SWPTABLE entries are marked in-transit, and a read operation is initiated for the missing page. The user is put in PAGEWAIT and control returns to the job scheduler (DISPATCH).

If the page was originally found to be in transit, the user is simply put in PAGEWAIT and control returned to DISPATCH.

When the page has been successfully read in, CP/67 removes the user from PAGEWAIT and adds a control program execution request block (CPEXBLOK) to the users CPRQUEST queue (in UTABLE). The next time DISPATCH examines these queues the CPEXBLOK will be executed. The page will thus be marked in-core and not-in-transit, thereby completing the paging operation.

2 Virtual I/O

When a user signs on, CP/67 examines his virtual machine configuration and creates the required virtual I/O blocks. For each multiplexor device, a new virtual multiplexor device block (MVDEBLOK) is created. It is chained to the previous MVDEBLOK and to a corresponding real multiplexor device block (MRDEBLOK). The address of the first MVDEBLOK is entered into the UTABLE.

For devices attached to selector channels, the list of virtual channel blocks is scanned for the channel address. If the channel address is not found, a virtual channel block (VCHBLOK), a virtual control unit block (VCUBLOK), and a virtual device block (VDEVBLOK) are created. The address of the first VCHBLOK is entered in the UTABLE. If the channel address is found, the chain of VCUBLOK's is scanned for the virtual control unit address. If not found, a VCUBLOK and a VDEVBLOK are created. The address of the first VCUBLOK is entered into the VCHBLOK. If the control unit address is found, only a VDEVBLOK need be created. The address of the first VDEVBLOK is entered in the VCUBLOK. Each VDEVBLOK is also chained to its corresponding real device block (RDEVBLOK).

Each channel, control unit, or device block is chained to the previous such block. The resulting relationship between virtual and real I/O blocks is illustrated in Fig. A.2.

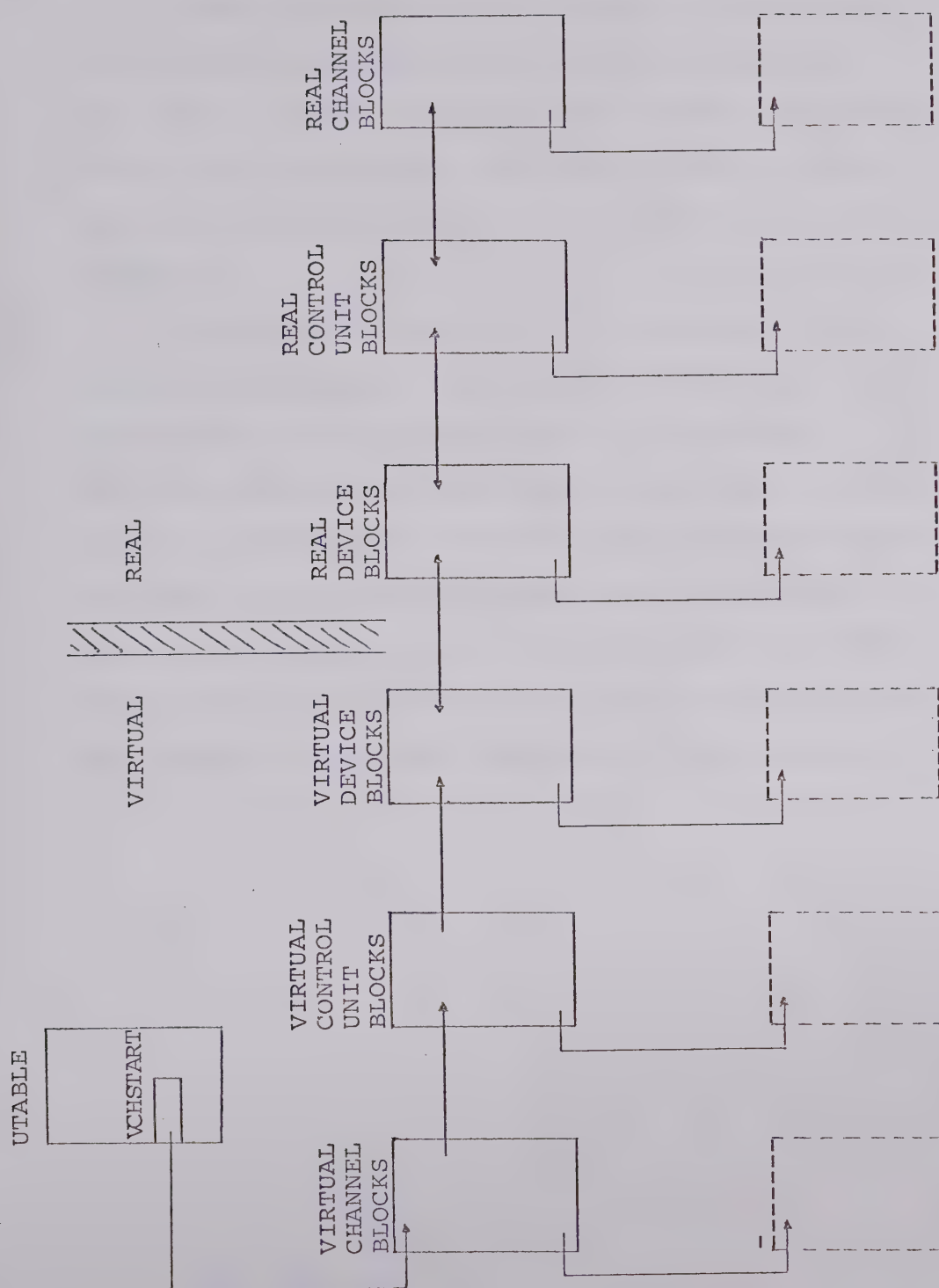


Fig. A.2 Virtual-Real I/O Blocks

When a virtual machine issues an I/O command, CP/67 will take control and, using the chains of I/O blocks, it will translate the virtual I/O command into a real I/O command. The user is put in IOWAIT and the real I/O is initiated. Control is returned to DISPATCH.

Conversely, when a real I/O interrupt occurs, the real interrupt is translated back into the virtual I/O blocks of the user for which it is intended. The user is removed from IOWAIT and an interrupt is marked PENDING in his UTABLE. The next time DISPATCH examines this user, the pending interrupt will be reflected to the virtual machine in its channel status word (CSW) and its PSW. The next time that user is chosen to run, the virtual machine will then handle its interrupt.

APPENDIX B

DETAILED DESCRIPTION OF DISPATCH

Figures B.1 and B.2 give a detailed flowchart illustration of the operation of DISPATCH. The six "choices" refer to the six criteria by which DISPATCH chooses the next user to run. They are described in Chapter IV (Sec. 4.2.1.2), and are outlined as follows:

- 1) RUNUSER, if runnable;
- 2) first NEXTUSER found in Q_1 and is runnable;
- 3) first NEXTUSER found in Q_1 -WAIT, and runnable;
- 4) first NEXTUSER found in Q_2 -WAIT, of highest priority, and runnable;
- 5) first NEXTUSER found in Q_2 , CLASS 1, of highest priority, and runnable;
- 6) first NEXTUSER found in Q_2 , CLASS 2, of highest priority, and runnable.

It should be noticed that the operation of DISPATCH involves two major loops in which each user, in turn, is examined.

1 Virtual Environment

The first loop (LOOP1) involves the maintenance of each user's virtual environment. Note that RUNUSER

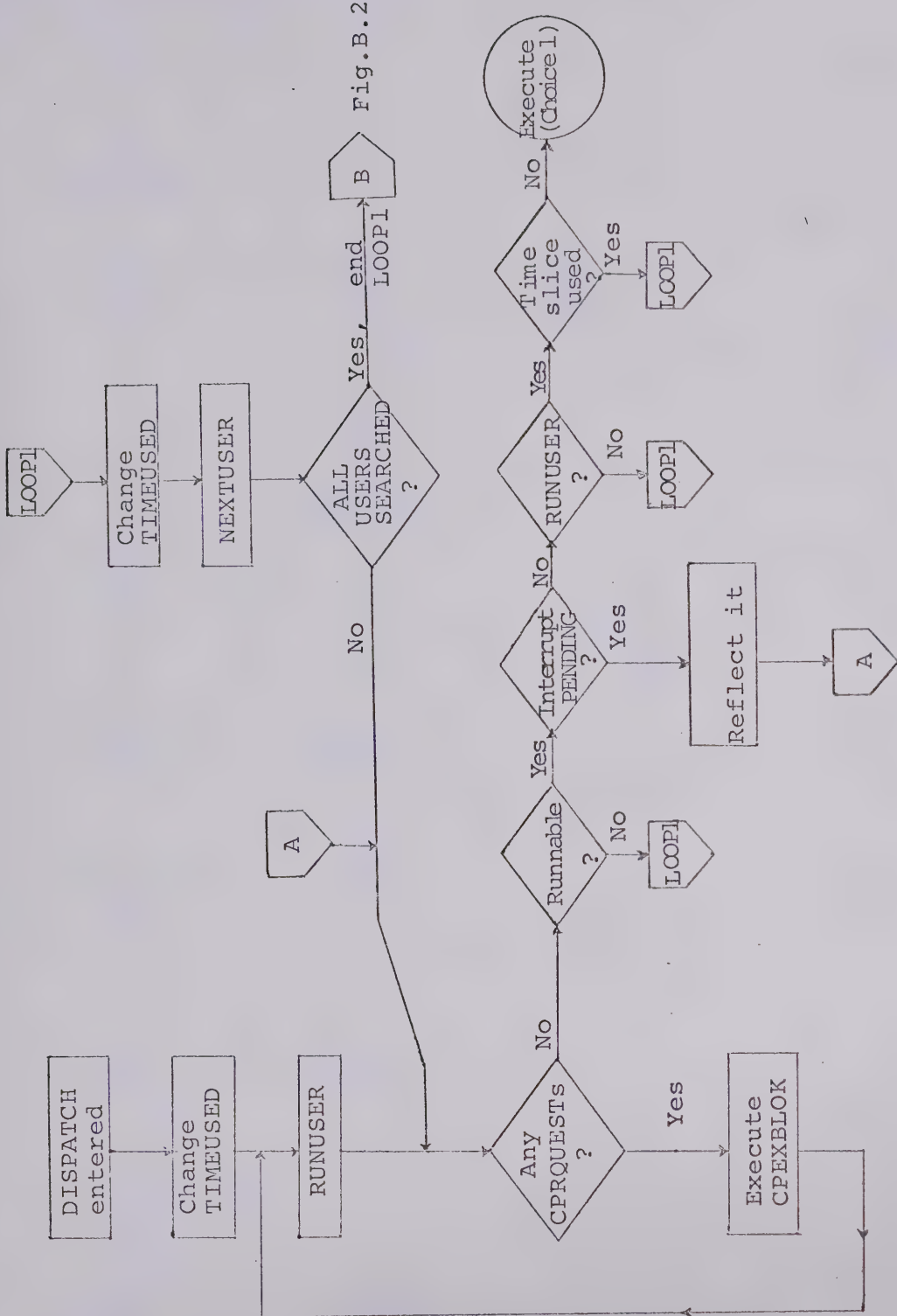
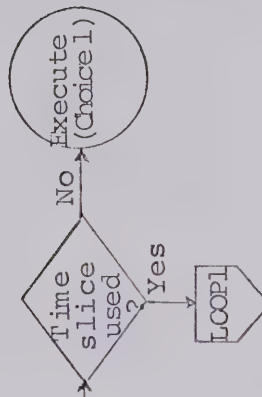


Fig. B.1 DISPATCH - LOOP1

Fig. B.2



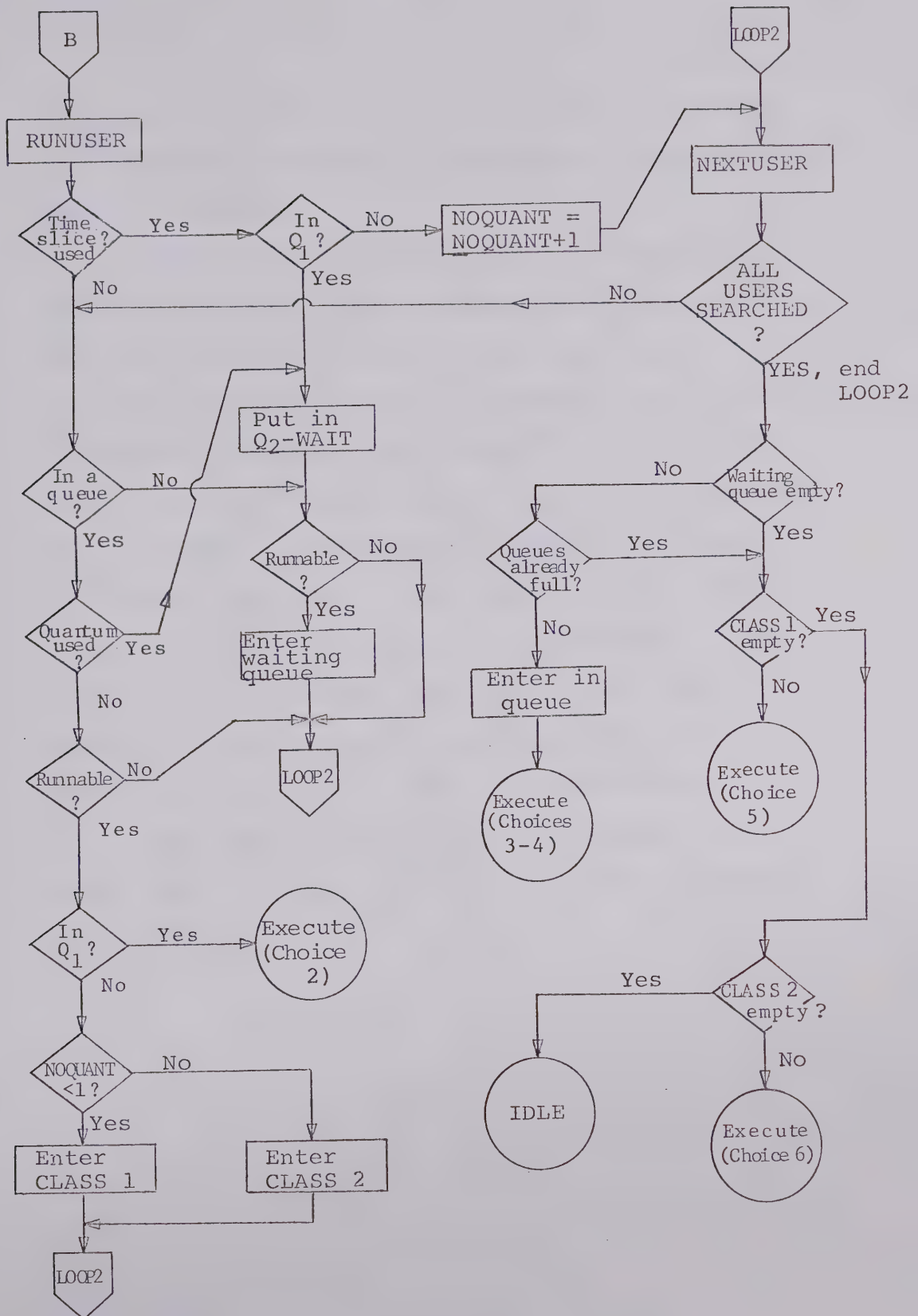


Fig. B.2 DISPATCH - LOOP2

is the first to be examined, and if he is chosen to run immediately (Choice 1) examination of the other users is preempted.

LOOP1 first examines the user's CPRREQUEST queue. If it is not empty, the top CPEXBLOK is removed and control transferred to it. After execution is completed, control returns to DISPATCH. The process is repeated until the users CPRREQUEST queue is empty. LOOP1 then examines the user to see if he is runnable. If not runnable, examination of NEXTUSER is begun. If runnable, the user is next examined for a pending interrupt. If one is found, it is reflected to the virtual machine and the user enters LOOP1 again. The process is repeated until no pending interrupt is found. Examination then begins on the NEXTUSER.

Each user is charged for the TIMEUSED in this loop. When all users have been examined, the second loop is begun.

2 Queue Maintenance

The second loop in DISPATCH (LOOP2) involves the up-to-date maintenance of the user queues (Q_1 and Q_2). Note again that the first user found who is in Q_1 and runnable is given control immediately (Choice 2), and further examination of the queues is preempted.

The management of Q_1 and Q_2 , illustrated in Fig. B.3, can be described as follows.

- 1) A user enters Q_1 -WAIT (waiting to enter Q_1) at the beginning of a request from the terminal.
 - 2) A user enters Q_1 only from Q_1 -WAIT, provided Q_1 is not already full. When a user enters Q_1 he is given a .4 sec. quantum or time limit.
 - 3) A user enters Q_2 -WAIT (waiting to enter Q_2) from either Q_1 or Q_2 .
 - a) A user will be removed from Q_1 under two conditions;
 - i) if he uses up his .4 sec. quantum of TIMEUSED in Q_1 , or
 - ii) if he at any time uses a full 50 msec. time-slice without an interruption.
- A user entering Q_2 from Q_1 starts with top priority.

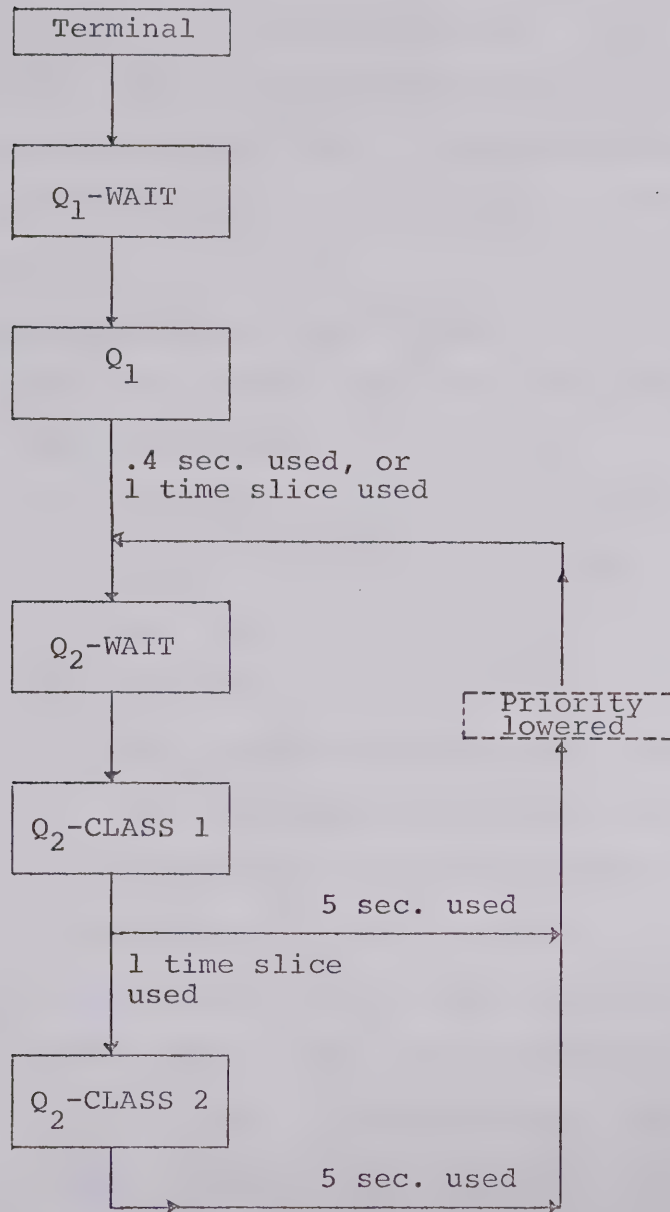


Fig. B.3 Queue Management

- b) A user will be removed from Q_2 when he uses up his 5 sec. quantum in Q_2 .
- 4) A user enters Q_2 only on the condition that it is not already full. A user leaving Q_2 , CLASS 1 or 2, has his priority lowered each time.
 - a) A user enters CLASS 1 only from Q_2 -WAIT. He leaves CLASS 1 under the following two conditions;
 - i) if he uses up his 5 sec. quantum, in which case he enters Q_2 -WAIT again, or
 - ii) if he at any time uses a full 50 msec. time-slice uninterrupted, in which case his NOQUANT is incremented by one and he automatically enters CLASS 2.
 - b) A user enters CLASS 2 only from CLASS 1. A user leaves CLASS 2 when he uses up his 5 sec. quantum. When he enters CLASS 2, a user retains the priority with which he just left CLASS 1.

The priority scheme within Q_2 is of a "bubble" nature. When a user's priority is lowered, it is made the lowest priority in Q_2 (rather than lowering it by

one notch). Hence all other users in Q_2 are effectively raised in priority. This is to prevent a long job from working itself so deep into Q_2 that it may never escape.

Note in Fig. B.2 that during LOOP2, a wait queue is maintained of all users waiting to enter a queue (Q_1 or Q_2). Users found waiting are entered into this wait queue according to the priority scheme given by Choices 3 and 4.

When LOOP2 is completed (i.e. all queues updated), DISPATCH is ready to select the next user to run (Choices 3-6). If no user is found in either the wait queue, CLASS 1 queue or CLASS 2 queue, then DISPATCH will enter CP/67 into the wait state.

The time used in LOOP2 is charged as OVERHEAD to the system. The total time spent in DISPATCH plus the time spent by CP/67 in handling the interrupt leading to DISPATCH's entry is accumulated as CPTIME. A time chart of all time charged by DISPATCH is given in Chapter IV, Fig. 4.1.

APPENDIX C

THE SIMULATION PROGRAM

Listing of all programs involved in this project may be obtained from the Department of Computing Science Library at the U of A. This appendix is intended to provide the general information needed to operate the simulation program in particular. The appendix and the GPSS program are complementary and should be used as supplements to each other.

The overall logic of the simulation program is described and illustrated in Chapter V (Fig. 5.2 and 5.3). The main concern of the program is to model the logic of DISPATCH. Appendix B gives a detailed description of DISPATCH from which this portion of the program logic is derived.

1 Model Input

Input to the model is described in terms of the number and type of users. To operate the model, then, it is first required to specify:

- a) the number of normal users (X1)
- b) the number of interactive users (X2)
- c) the number of heavy users (X3).

For each of these three types of users there is defined:

- a) a request size distribution (FN1-3)

- b) a think time distribution (FN4-6)
- c) an interrupt arrival rate distribution (FN7-9).

Two additional distributions are defined which are used by all three types of users to generate the interrupts associated with each request. These are:

- a) an interrupt wait distribution (FN10)
- b) a distribution giving the number of interrupts as a function of the size of the request (FN11).

These eleven GPSS "functions" (FN) may be used as they are presently defined or may be changed to any desired distribution. Functions 1 through 10 are used in conjunction with the random number generators of GPSS. Since there are eight such random number generators, these too may be altered to obtain variations in the generated input.

2 System Representation

Three constant values are used to represent the time used by CP/67 in the operation of the model. As described in Chapter VI, Sec. 1.2, they are:

- a) the time required to handle each interrupt
- b) the time used within DISPATCH which can be charged to an individual user (i.e. LOOP1)
- c) the time used within DISPATCH which cannot be charged to any individual user (i.e. LOOP2).

The figures at present are average values estimated from the data. However these figures are flexible and could be changed. For instance, (a) could be made a function of the type of interrupt if such information was available.

The operation of the system is also reflected by:

- a) the multiplication factor used to determine the number of interrupts as a function of the number of users (V6)
- b) the multiplication factor used to determine the interrupt wait time as a function of the number of users (V12).

These factors are currently set at $(\frac{N}{15})^2$, where N is the number of users, but can be altered as desired (see discussion in Chapter VI, Sec. 2.2.2).

3 DISPATCH parameters

The following parameters of DISPATCH must be specified at the beginning of each run:

- a) timeslice (X85)
- b) Q_2 quantum (msec.) (X86)
- c) Q_1 quantum (msec.) (X87)
- d) maximum no. of users allowed in Q_2 (X88)
- e) maximum no. of users allowed in Q_1 (X89).

The final two values which must be specified are:

- a) the warmup time (sec.) (X90)
- b) the running time (sec.) (X91).

Together with the logic of the system, as modeled in the program, complete representation of the system can be achieved by specifying the above set of distributions, parameters, etc.

B29962